Nuno Henrique Costa Figueiredo Chaves Marujo

SISTEMA DISTRIBUÍDO DE AQUISIÇÃO E CONTROLO BASEADO EM LINGUAGEM JAVA

Nuno Henrique Costa Figueiredo Chaves Marujo

SISTEMA DISTRIBUÍDO DE AQUISIÇÃO E CONTROLO BASEADO EM LINGUAGEM JAVA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. Alexandre Mota, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho aos meus pais, por todo o apoio, amizade e confiança ao longo de todos estes anos. Sem eles nada disto seria possível.

O júri

Presidente

Prof. Dr. José Alberto Gouveia Fonseca
Professor Associado da Universidade de Aveiro

Vogais

Prof. Dr. José António Barros Vieira
Professor Adjunto do Instituto Politécnico de Castelo Branco

Prof. Dr. Alexandre Manuel Moutela Nunes da Mota (Orientador)
Professor Associado da Universidade de Aveiro

Agradecimentos

Ao longo desta dissertação obtive muitos contributos de várias pessoas, pelos quais estou grato e que não devem ser esquecidos. Em especial gostaria de agradecer:

Ao meu orientador, Prof. Alexandre Mota, pelo seu constante apoio, pelo entusiasmo que sempre transmitiu e pela sua constante boa disposição.

Aos meus colegas orientandos, Luís Farinha, José Santos e Rui Sancho pelos momentos que partilhámos ao longo deste ano, pelo espírito de inter-ajuda, constante apoio e preciosa amizade.

Aos meus pais e irmãos pelo seu constante apoio. Por acreditarem em mim e pela força que transmitiram.

Aos meus colegas de casa, Luís Oliveira e Pedro Brochado, por toda a sua amizade e boas recordações ao longo destes anos.

Aos meus amigos e colegas, Jorge Rodrigues e Margarida Fernandes, por todo o apoio, amizade e óptimos momentos que passámos juntos que sempre ajudaram a relaxar em momentos de maior pressão e stress.

Aos meus colegas do laboratório 234, João Lima, Pedro Henriques, Nuno Duarte, Abílio Neves e Hugo Barreira pelo excelente clima de trabalho, por toda a ajuda e amizade.

A todos os meus restantes amigos e família cujo envolvimento não foi tão directo mas que não devem, igualmente, ser esquecidos. O meu muito obrigado por todo o apoio e amizade ao longo destes anos.

Palavras-chave Java, controlo distribuído, *Ethernet*, *sockets*, TCP/IP, *gateway*

Resumo

Actualmente a linguagem Java é mundialmente utilizada e cada vez mais são as áreas onde pode ser encontrada. A sua popularidade deve-se principalmente à facilidade que proporciona na criação de aplicações distribuídas.

Esta linguagem apresenta restrições ao nível do determinismo temporal, o que a tem afastado da área do controlo. No entanto estas limitações podem ser desprezadas quando a dinâmica dos sistemas a controlar é lenta.

Sob este pressuposto são propostas, no âmbito desta dissertação, várias filosofias de criação de malhas de controlo distribuídas implementadas em linguagem Java em plataformas do tipo PC.

São, adicionalmente, apresentados alguns resultados da aplicação destas filosofias em alguns sistemas.

Keywords Java, distributed control, *Ethernet*, *sockets*, TCP/IP, *gateway*

Abstract

Currently the Java language is widely used and the areas where it can be found are increasing. Its popularity is mainly due to the ease it provides in creating distributed applications.
This language has some restrictions on the level of temporal determinism, which kept it away from the control area.
However these limitations can be neglected when the dynamics of the control systems are slow.
Under this assumption several philosophies are proposed, in the context of this dissertation, to create distributed control loops implemented using Java language in the standard PC platforms. Results of application of these philosophies in some systems are also presented.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Este trabalho trata do estudo e implementação de sistemas de aquisição e controlo baseados em linguagem Java. É abordada a implementação de malhas de controlo distribuídas baseadas em nós do tipo PC interligados por redes de comunicação baseadas em tecnologia *Ethernet*. Pretende-se tomar partido da facilidade de implementação de aplicações distribuídas que a linguagem Java proporciona bem como aproveitar todas as suas potencialidades de interface. Na interface com sensores e actuadores é tida em conta a utilização de sensores interligados por barramentos 1-Wire [1], bem como equipamentos de instrumentação com suporte à norma SCPI [2], entre outros.

Neste estudo são tidas em conta o tipo de limitações associadas ao uso destas tecnologias, pelo que este será focado no controlo de sistemas com características satisfatórias para o uso das mesmas. Isto implica o controlo de sistemas com uma dinâmica lenta tal como encontrado, por exemplo, em aplicações do âmbito da engenharia agrícola ou em controlo de temperatura.

1.1 Estado da arte

Ao longo da História, a curiosidade e ambição Humanas têm sido impulsionadoras de grandes avanços tecnológicos. A tecnologia nasceu como fruto da necessidade do Homem de criar meios ou utensílios por forma a controlar o seu meio circundante. As mais simples ferramentas, feitas a partir dos recursos naturais mais básicos, a invenção da roda ou a descoberta do fogo, entre muitos outros, tiveram um papel determinante para o culminar daquilo a que hoje designamos por sociedade moderna, salientando-se, talvez mais, o papel deste último como fonte de energia, para criar e sustentar, directa ou indirectamente, quase tudo o que conhecemos de tecnologia moderna.

Com a evolução da tecnologia deu-se naturalmente um aumento da sua complexi-

dade, face à também crescente complexidade dos sistemas a controlar. A vontade de controlar sistemas cujo comportamento era de algum modo complexo, pouco intuitivo ou muito difícil de prever, conduziu à necessidade de criar métodos de análise rigorosos e formalizados, um novo ramo de estudo mais assente no conhecimento científico. Surge, então, a designada Teoria de Controlo, um ramo interdisciplinar com fortes bases na matemática e engenharia.

A Teoria de Controlo é, hoje em dia (após o seu impressionante crescimento maioritariamente a partir do século XIX), um ramo bastante robusto e amadurecido, podendo observar-se os seus frutos, nos dias que correm, nos mais variados sistemas que nos rodeiam.

Numa perspectiva mais formal, um sistema de controlo consiste em subsistemas e processos, assemblados com o propósito de controlar as saídas dos mesmos [3].

Com o aparecimento dos computadores digitais e a sua expansão explosiva deu-se o aparecimento de um novo ramo na Teoria de Controlo designado por Controlo Digital. Neste tipo de controlo, o computador digital assume um papel de relevo como controlador do sistema. Dado que os computadores digitais se tratam de sistemas discretos, tornou-se necessário modificar a forma como se analisavam os sistemas. Estes deixaram de ser descritos no domínio contínuo passando a ser tratados no domínio discreto. Tornou-se assim comum o uso de conversores analógico-digitais (ADC) e digitais-analógico (DAC), precedendo a ligação do computador digital à entrada e saída dos sistemas, respectivamente, tal como pode ser visualizado na Figura 1.1 [4]. Apesar da necessidade de todas estas alterações, comparativamente ao controlo no domínio contínuo, o Controlo Digital possui muitas vantagens que facilmente suprimem as desvantagens que possa apresentar. A elevada e contínua descida de preço que os computadores digitais sofreram, desde a criação do primeiro exemplar no início dos anos 40, tornou esta tecnologia bastante apetecível. Aliado a isto, a sua flexibilidade de programação e reconfiguração através do *software*, face às modificações feitas em *hardware*, tornam-no num sistema muito mais prático. A sua insensibilidade ao meio ambiente confere-lhe fiabilidade, garantindo reproducibilidade de resultados. Outra grande vantagem que apresenta é a sua escalabilidade, principalmente perante as exigências de uma sociedade cada vez mais industrializada. Não se devendo deixar de referir a sua adaptatividade, que se revela também um factor chave, possibilitando que os parâmetros do sistema possam ser alterados ao longo do tempo.

O desenvolvimento dos computadores digitais propiciou, de igual modo, a evolução das redes. Os computadores evoluíram ao ponto em que parecia que o passo mais lógico a tomar seria o de interligá-los por forma a que os mesmos pudessem usufruir não só

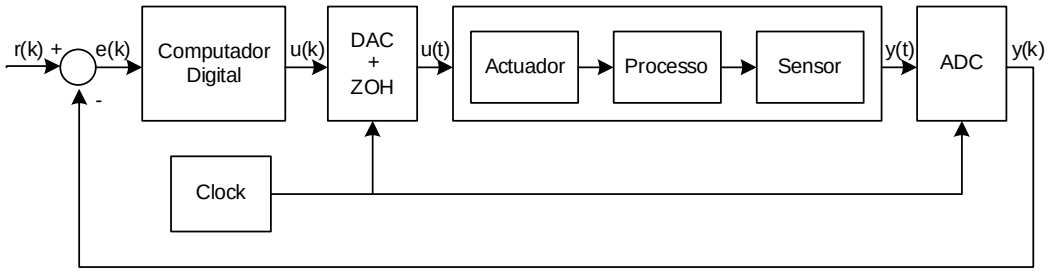


Figura 1.1: Sistema de controlo baseado em computador digital

dos seus recursos locais, mas também dos recursos existentes nas suas redes, podendo ser vistos como recursos remotos. A interligação de várias entidades computacionais através de uma rede permitiu o aparecimento de aplicações distribuídas, face à maioria das aplicações existentes até então, que eram tendencialmente monolíticas. Apesar dos primeiros passos na direcção dos sistemas distribuídos terem ocorrido na década de 70, estes apareceram efectivamente nos anos 90 devido à excelente relação preço/qualidade que os sistemas baseados em microprocessador apresentavam e à maturidade que as redes tinham atingido. Os sistemas distribuídos apresentam um maior aproveitamento de recursos servindo-se de recursos locais e remotos. São sistemas facilmente remodeláveis e expansíveis quando bem projectados, demonstrando assim grande flexibilidade e escalabilidade. Tornam a modularização mais fácil e intuitiva e viabilizam o projecto de interfaces de controlo e comando que se desprendem da necessidade ou conveniência de proximidade física. São, por si só, um tipo de processamento paralelo, que optimiza a gestão de tempo e de recursos. Apresentam baixo custo e o facto de serem constituídos por múltiplas unidades de armazenamento e processamento confere-lhes maior fiabilidade e tolerância a falhas [5]. Muitas destas propriedades são muito importantes na implementação de sistemas que necessitem de garantias de tempo-real.

De todos os tipos de redes que surgiram até hoje, são de salientar as redes do tipo *Ethernet*. Estas sofreram, de longe, uma muito maior aceitação do que as restantes. A *Ethernet* foi originalmente criada na empresa Xerox PARC, entre os anos de 1973 e 1975. A partir daí, deu-se um esforço, pela parte dos seus criadores, no sentido de aliar a *Ethernet* ao uso de computadores pessoais e redes locais (*Local Area Networks* ou LANs). Como resultado deste esforço surgiu a empresa 3Com, que mais tarde conseguiu convencer empresas como a DEC, a Intel e a Xerox a trabalhar como equipa no sentido de tornar a *Ethernet* num padrão. Este padrão foi publicado em 30 de Setembro de 1980 competindo contra grandes sistemas da altura, como o *Token Ring* e ARCNET. A padronização levou pouco tempo a conduzir a uma proliferação explosiva desta tecnologia arrasando os seus competidores. Depressa uma gama alargada de

produtos *Ethernet* estavam disponíveis e, por serem compatíveis com a grande maioria dos sistemas após a padronização, depressa também o seu preço reduziu bastante. A descida de preço e a evolução das capacidades desta tecnologia acompanharam-se lado a lado, levando a que fosse cada vez mais utilizada. Originalmente, a velocidade de transmissão padrão era de 10 Mbit/s, demonstrando-se bastante rápida e suficiente durante muitos anos. Em 1995 foi aumentada para 100 Mbit/s. Esta tecnologia foi denominada de *Fast Ethernet*. Mais tarde, em 1998, numa tentativa de antecipar a crescente demanda de velocidade, foi criada a *Gigabit Ethernet*, apresentando uma espantosa velocidade de transmissão de 1 Gbit/s. Um ponto importante nesta evolução foi a permanência de compatibilidade com as versões antigas que se encontrou presente nos novos produtos. Isto garantiu à *Ethernet* grande flexibilidade, facilidade de transição e escalabilidade. A utilização, desde 1987, de ligações do tipo par-entrançado permitiu criar também sistemas bem estruturados, simples e muito mais fiáveis. Hoje em dia, a *Ethernet* é uma tecnologia global de grande importância, tendo proliferado de tal forma que se torna muito difícil encontrar áreas onde não se aplique. *Routers*, *bridges*, *repeaters* e *switches*, entre outros equipamentos, deram grande potencial às redes, interligando todo o Mundo, fazendo a informação fluir rápida e eficazmente, e transformando-o cada vez mais numa aldeia global [6].

Um tipo particular de aplicações distribuídas que surgiu foram as aplicações de controlo distribuídas. Tal como referido em [7], um sistema de controlo distribuído é composto por sensores, actuadores e controladores cujas operações são distribuídas por diferentes localizações geográficas e coordenadas pela informação trocada nas redes de comunicações. Estes sistemas têm sido alvo de grande aceitação e procura, principalmente pela parte dos sectores industriais e militares.

Actualmente existe uma forte tendência à partição das aplicações, inerente ao desenvolvimento tecnológico, que tem propiciado o desenvolvimento de linguagens como o Java. Esta é uma linguagem inovadora em vários aspectos, com grande potencial, que tem sido alvo de um desenvolvimento admirável. O seu objectivo principal é tornar a criação de aplicações distribuídas sobre redes o mais simples e portável possível. Esta tem sido mundialmente adoptada em várias áreas, não o sendo, no entanto, com tanta frequência, em sistemas de controlo distribuído. Isto deve-se, principalmente, ao indeterminismo temporal que apresenta, indesejável em sistemas com requisitos temporais muito rígidos. No entanto, em sistemas mais flexíveis, esta linguagem já tem sido aplicada para efeitos de controlo e aquisição de dados. São exemplos disto aplicações que usam a 1-Wire® API para Java™, no uso de barramentos 1-Wire, ou então a plataforma JDDAC (*Java Distributed Data Acquisiton and Control*) [8].

A plataforma JDDAC é destinada à criação de redes de sensores e sistemas de

aquisição de dados. Esta plataforma foi desenvolvida em Java, com base nas normas de IEEE 1452 para transdutores inteligentes. A 1-Wire® API para Java™ é uma API desenvolvida em Java para acessos a barramentos do tipo 1-Wire. Este tipo de barramentos são descritos com mais detalhe no Capítulo 2 desta dissertação.

Nos dias que correm computadores, redes e controlo digital estão presentes em todo o lado, existindo inúmeras áreas onde estes se aplicam, desde a indústria automóvel, aviónica, domótica, entre muitos outros. Vivemos cada vez mais numa era de automação e controlo automático, onde estes se tornaram uma constante. Encarados quase como dados adquiridos, facilitam-nos a vida do dia-a-dia, ajudando a produzir mais e melhor. Muitas vezes estão escondidos ao primeiro olhar, mas estão bem difundidos e dificilmente imaginaríamos o Mundo que conhecemos sem a sua presença.

1.2 Motivação

Os sistemas de controlo têm um papel muito importante na tecnologia actual. Permitem, actualmente, o controlo de uma série de sistemas de elevada complexidade, que não parecia possível há alguns anos atrás. Um tipo particular de sistemas de controlo são os sistemas de controlo distribuído que se tornaram, igualmente, muito populares. Têm sido bastante utilizados, não se resumindo, o seu âmbito, à área industrial ou militar. É cada vez mais frequente a sua presença nos mais variados sistemas presentes no quotidiano. É exemplo disto o progresso e aceitação de que têm vindo a ser alvo algumas tecnologias nas áreas da domótica e das telecomunicações. A noção de virtualização dos sistemas é cada vez mais uma constante. As pessoas desejam mobilidade, sem que isso inviabilize o seu acesso aos mesmos. Popularizou-se, portanto, a partição e distribuição dos sistemas.

A evolução das redes e dos sistemas computacionais têm alargado os horizontes tecnológicos da actualidade. A *Internet* sofreu uma grande evolução, e actualmente podemos aceder-lhe virtualmente em qualquer lado. A tendência é a de, cada vez mais, criar sistemas baseados e distribuídos sobre redes e muitos destes são feitos sobre a *Internet*.

Uma linguagem de programação que surgiu, para servir este propósito, foi a linguagem Java. Esta linguagem é actualmente mundialmente utilizada, principalmente em aplicações distribuídas sobre redes. É frequente encontrá-la em telemóveis, e na *Internet*, entre outros, servindo os mais diversos propósitos.

A sua utilização em sistemas de controlo distribuído não é, no entanto, muito frequente, dado o indeterminismo temporal que apresenta. Contudo, esta é uma lingua-

gem cujas potencialidades têm propiciado muito estudo e desenvolvimento em torno de si mesma. Está, actualmente, bem desenvolvida, mas possui ainda muitos aspectos por onde promete evoluir. Isto leva a que, cada vez mais, haja um esforço no sentido de adaptá-la, por forma a que esta consiga cumprir os requisitos suficientes para poder ser usada na implementação de um maior leque de sistemas de controlo distribuído.

Num nicho mais restrito, dos sistemas de controlo distribuído, cujos requisitos são menos rígidos, esta já pode, no seu estado actual, ser utilizada, revelando potencialidades que a tornam, muitas vezes, uma escolha acertada.

Algumas destas potencialidades serão, mais tarde, referidas no capítulo 2 desta dissertação.

1.3 Pressupostos

Ao longo desta dissertação, todos os testes e soluções apresentadas foram executados sob uma série de pressupostos sem os quais não teriam viabilidade.

- Os sistemas a controlar possuem requisitos de tempo-real do tipo *soft*. Esta definição será apresentada no capítulo 2.

- Estes mesmos sistemas possuem uma dinâmica suficientemente lenta, que garante que o indeterminismo temporal que a Linguagem Java apresenta pode ser desprezado, viabilizando assim o seu uso na implementação de uma malha de controlo.

- A dinâmica lenta dos sistemas a controlar possibilita também o uso de redes de comunicação baseadas em tecnologia *Ethernet*, dado que, deste modo, o indeterminismo temporal que esta tecnologia apresenta tem efeitos que podem ser desprezados.

- Os períodos de amostragem são suficientemente grandes, viabilizando o uso de sensores interligados por barramento 1-Wire, bem como o uso de equipamentos de instrumentação com suporte à norma SCPI.

1.4 Estrutura da dissertação

Esta dissertação está organizada em cinco capítulos. No capítulo 2 são apresentados alguns conceitos introdutórios relativamente a tecnologias utilizadas no âmbito desta

dissertação. É efectuada uma descrição dos mesmos apresentando aspectos relevantes na sua caracterização.

No capítulo 3 são apresentadas algumas estruturas possíveis para a criação de sistemas de controlo baseados na utilização conjunto da linguagem Java e programas de computação numérica. São mencionadas as vantagens de cada diferente tipo de estruturação, assim como alguns algoritmos exemplificativos da sua utilização.

No capítulo 4 são apresentados alguns exemplos de aplicação dos conceitos expostos no capítulo 3. É demonstrado o controlo de um processo térmico usando MATLAB e Java, e posteriormente usando Scilab e Java. Será, adicionalmente, demonstrado um sistema de aquisição baseado em barramentos do tipo 1-Wire.

No capítulo 5 são feitas as conclusões relativas ao trabalho desenvolvido nesta dissertação e são apresentadas algumas sugestões relativamente a trabalho futuro.

Finalizando este documento surgem quatro anexos. O primeiro caracterizando o sistema de controlo de processo térmico (módulo PCT 13), o segundo demonstrando os passos para inclusão de classes no MATLAB, o terceiro fazendo uma descrição da estrutura da API Java desenvolvida no âmbito desta dissertação e o quarto demonstrando o *hardware* desenvolvido.

Capítulo 2

Conceitos Fundamentais

Este capítulo aborda alguns conceitos introdutórios considerados fundamentais na caracterização ou diferenciação de tecnologias utilizadas. Os conceitos são expostos tentando sempre evidenciar as suas principais vantagens, referindo algumas das suas limitações e identificando os casos meritórios de uso.

2.1 Sistemas de controlo distribuído

Nos dias que correm, os sistemas de controlo distribuído são cada vez mais usuais. São inúmeros os seus campos de aplicação, desde a domótica e automação até à exploração espacial e terrestre, entre outros.

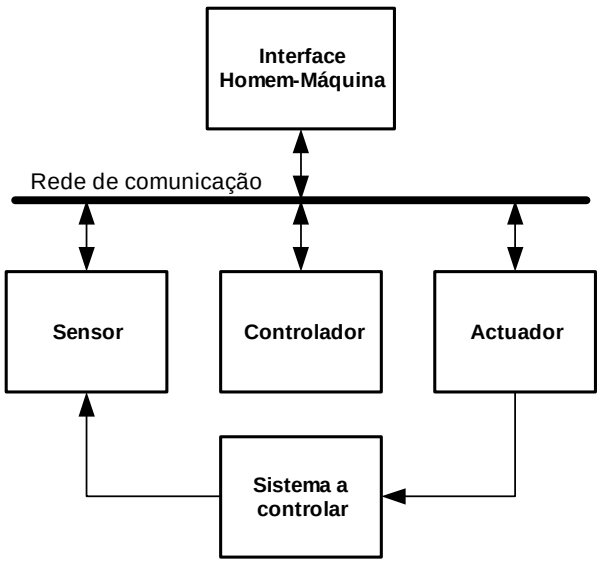


Figura 2.1: Sistema de controlo distribuído

Os sistemas de controlo distribuído são sistemas cuja malha de controlo é fechada

por uma rede de acesso comum aos nós. Estes nós são, maioritariamente, sensores, controladores e actuadores. Adicionalmente é frequente a existência de interfaces Homem-Máquina para efeitos de parametrização do sistema, diagnóstico e monitorização. Uma configuração comum de um sistema de controlo distribuído é aquela apresentada na Figura 2.1. Quanto às redes de comunicação usadas em sistemas de controlo distribuído estas são, usualmente, barramentos de campo.

2.1.1 Sensor

Sensor é qualquer dispositivo electrónico sensível a uma quantidade física e capaz de a converter num sinal mensurável por um utilizador ou um instrumento. Existem uma série de sensores dos mais variados tipos, utilizados em todo o Mundo, como por exemplo, sensores de temperatura, humidade, luminosidade, pressão, entre muitos outros.

Os sensores são um componente importante da malha de controlo, visto que são estes que permitem que o controlador conheça o estado do sistema. Sem estes qualquer tipo de controlo por realimentação seria impossível de realizar.

Numa malha de controlo distribuída, os sensores estão frequentemente num bloco separado, inclusivamente bastante distanciados geograficamente dos restantes blocos do sistema.

2.1.2 Controlador

Controlador é o elemento da malha de controlo responsável por processar e analisar os dados adquiridos dos sensores, indicando seguidamente ao actuador a forma como deverá interagir com o sistema. Os dados são processados segundo um algoritmo de controlo que, consoante o sistema em causa, pode variar muito em grau de complexidade.

Numa malha de controlo distribuída este é, tipicamente, o bloco com maior carga de processamento, propriedade que varia igualmente com o algoritmo de controlo em causa. Isto leva a que este requeira, tipicamente, uma entidade computacional mais potente do que as restantes presentes no sistema. No entanto, este pode eventualmente estar situado na mesma entidade computacional que os sensores ou actuadores. Esta escolha depende muito do sistema em causa e dos objectivos a atingir.

2.1.3 Actuador

Actuador é o elemento da malha de controlo que interage com o sistema, com o intuito de controlar o seu comportamento, por forma a que este seja o mais próximo possível do que se estipulou que deveria ser. O actuador actua de acordo com o indicado pelo controlador.

O actuador é, tipicamente, um mecanismo mecânico, pneumático ou eléctrico que controla um determinado movimento, afectando, de algum modo, o estado do sistema. A sua localização física é, portanto, muito importante, tendo muitas vezes de seguir requisitos muito específicos. Isto leva a que, em sistemas de controlo distribuído, este se encontre muitas vezes num bloco distinto, distanciado dos demais.

2.1.4 Interface Homem-Máquina

A interface Homem-Máquina é um elemento da malha de controlo que visa proporcionar um acesso ao utilizador, para efeitos de parametrização dos sistemas, diagnóstico ou monitorização dos mesmos. Tem, muitas vezes, necessidade de ser colocada num local remoto do sistema a controlar, como, por exemplo, em fábricas, onde são muitas vezes centralizadas em centros de comando, para uma gestão global simplificada. Por esta razão, num sistema de controlo distribuído, as *interfaces* ocupam, frequentemente, lugar numa entidade computacional distinta dos restantes elementos.

2.1.5 Rede de comunicação

Um elemento muito importante das redes de controlo distribuído é a rede de comunicação. Tendo em conta que grande parte dos sistemas de controlo distribuído estão inseridos na indústria a rede utilizada é, usualmente, um barramento de campo.

Barramento de campo é uma rede de comunicação industrial, que visa ser usado no controlo de sistemas em tempo-real. É uma tecnologia que foi padronizada com o intuito de melhorar a performance dos sistemas acabando com as linhas analógicas de 4-20mA até então muito usadas. Esta tecnologia abrange vários protocolos para redes industriais como, por exemplo, o CAN, Profibus, WorldFIP, entre outros [9].

No entanto numa grande quantidade de sistemas de controlo distribuído, inclusive no âmbito industrial, é comum encontrarem-se redes de comunicação que não são barramentos de campo. Neste subconjunto de sistemas de controlo distribuído, os requisitos temporais e de segurança são normalmente menos rígidos. Consequentemente, a rede de comunicação que utilizam não necessita de possuir propriedades tão específicas, podendo, na grande maioria das vezes, ser usadas redes de propósito mais generalista.

Um exemplo recorrente deste tipo de situações pode ser encontrado em sistemas de controlo distribuído baseados em redes de comunicação do tipo *Ethernet*.

2.1.6 Principais vantagens

Os sistemas de controlo distribuído possuem várias vantagens relativamente ao controlo centralizado, que têm conduzido à sua frequente utilização. Podemos, de entre estas, realçar as seguintes [7, 10]:

Optimização de recursos - sendo um tipo de aplicação distribuída, implementa inerentemente um tipo de processamento paralelo. Optimiza, assim, a gestão de tempo e recursos.

Redução da complexidade - interligar sensores, controladores e actuadores através de uma rede conduz a uma estruturação mais modularizada e, consequentemente, menos complexa.

Facilidade de reconfiguração - a implementação sobre uma rede facilita reconfigurações no sistema, desde localização física ao papel de cada nó. Isto torna o sistema mais fácil de reaproveitar e actualizar.

Diagnóstico - em caso de falhas o diagnóstico torna-se mais simples podendo, frequentemente, ser efectuado remotamente. Isto reduz igualmente custos e a quantidade de mão-de-obra necessária.

Manutenção - a manutenção torna-se, na maioria das vezes, mais simples. Esta pode ser, muitas vezes, efectuada por acesso remoto. Isto reduz igualmente custos e quantidade de mão-de-obra necessária.

Partilha de dados - a interligação de elementos via uma rede proporciona inerentemente a facilidade de partilha de informação, tornando simples o cruzamento de informação da rede para tomar decisões a uma larga escala geográfica.

Redução de cablagem - o uso de uma rede partilhada elimina o uso de qualquer cablagem desnecessária.

Escalabilidade - torna fácil a adição de elementos, com baixo custo e pequenas mudanças de estruturação.

Interligação - interligam o ciberespaço e o espaço físico, tornando a execução de tarefas remotas facilmente acessível.

Fiabilidade - sendo constituídos por redes, várias entidades computacionais e várias unidades de armazenamento, tornam mais simples a recuperação em casos de corrupção ou perda de dados, aumentando assim a fiabilidade dos sistemas.

Tolerância a falhas - em caso de uma falha num nó, esta pode ser detectada, e nós com funções idênticas podem tomar a iniciativa de o substituir temporariamente, enquanto estes são reparados ou substituídos.

2.2 Sistemas de tempo-real

Segundo Motus e Rodd [11], tempo-real é uma propriedade dos sistemas computacionais que caracteriza a sua capacidade de estabelecer correspondências entre diferentes sistemas de medição e/ou contagem de tempo. Diz-se que um sistema é de tempo-real se servir, pelo menos, um requisito de tempo-real. Este conceito é, no entanto, muitas vezes mal interpretado. O senso comum tem tendência a enganar-nos frequentemente, levando-nos a acreditar que tempo-real significa rapidez. No entanto isto não corresponde à verdade. Um sistema diz-se trabalhar em tempo-real se for capaz de trabalhar ao ritmo imposto pela dinâmica do sistema a controlar. Esta dinâmica está, frequentemente, associada aos tempos de resposta de processos físicos, e nada tem a ver com o sistema computacional em si.

2.2.1 Requisitos de tempo-real

Todos os sistemas de tempo-real possuem requisitos, mas nem todos são do mesmo tipo ou têm o mesmo grau de rigidez. São classificados em três grupos: *soft*, *firm* e *hard*.

Soft

Os requisitos do tipo *soft* são tolerantes ao incumprimento de *deadlines*, nos quais a informação que chega após as mesmas ainda tem alguma utilidade, sofrendo no entanto um decréscimo com o passar do tempo, associado a uma perda de qualidade de serviço, tal como representado na Figura 2.2 (a). São exemplos disto os requisitos presentes em sistemas de aquisição de imagem para condução de pequenos robôs, entre outros.

Firm

Os requisitos do tipo *firm* são intolerantes ao incumprimento de *deadlines*, nos quais a informação que chega após as mesmas não tem qualquer utilidade. Tal como representado na Figura 2.2 (b). São exemplos disto os requisitos presentes em sistemas de leitores de DVD ou *stream* de vídeo, entre outros.

Hard

Os requisitos do tipo *hard* são igualmente intolerantes ao incumprimento de *deadlines*, nos quais a informação que chega após as mesmas não só não tem qualquer utilidade,

como pode adicionalmente ter consequências catastróficas. Tal como representado na Figura 2.2 (c). São exemplos disto os requisitos presentes em sistemas de aviónica, controlo de centrais nucleares, entre outros.

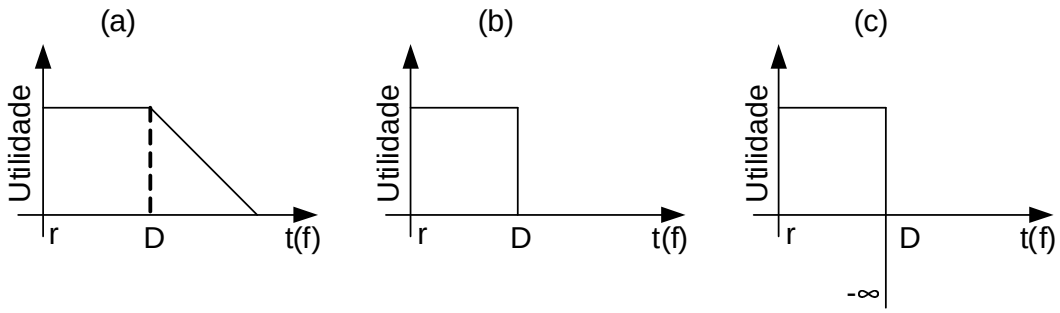


Figura 2.2: Tipos de requisitos de tempo-real

2.2.2 Tipos de sistemas de tempo-real

Existem dois tipos de sistemas de tempo-real. Estes são classificados como *soft* ou *hard* consoante o grau de tolerância que apresentam ao incumprimento de *deadlines*.

Soft

Os sistemas do tipo *soft* são aqueles que apresentam apenas requisitos temporais do tipo *soft* ou *firm*. Neste grupo englobam-se todos os sistemas cujas falhas não são consideradas perigosas.

Hard

Os sistemas do tipo *hard* são aqueles que apresentam, pelo menos, um requisito temporal do tipo *hard*. Neste grupo englobam-se todos os sistemas de tempo-real que de alguma forma são considerados de segurança crítica.

2.3 Redes *Ethernet*

As redes *Ethernet* são, de longe, o tipo de redes mais utilizadas a nível mundial. Estas são dominantes a nível de redes locais e do acesso à *Internet*. Actualmente, é comum existirem redes de fibra óptica a servir de *backbone*, enquanto que redes *Ethernet* são ligadas às mesmas, fornecendo acesso a redes locais, de média e pequena dimensão, como representado na Figura 2.3.

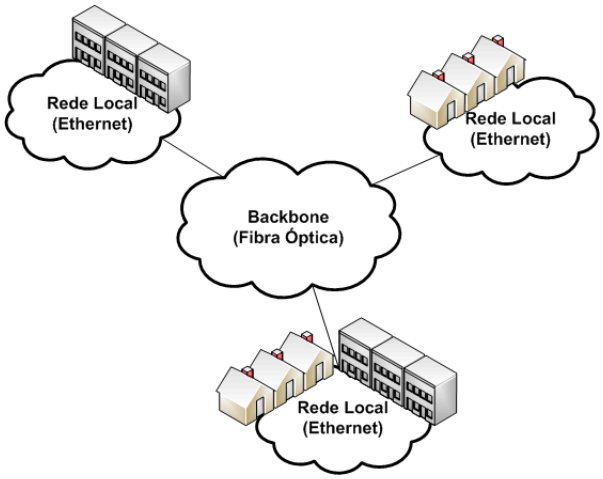


Figura 2.3: Rede de dimensão alargada

2.3.1 Principais vantagens

A tecnologia *Ethernet* é actualmente utilizada a nível global. A sua adopção conduziu a uma evolução da mesma, acrescentando vantagens na sua utilização e reforçando algumas há muito existentes. De entre elas, estão por exemplo [6]:

Custo - é uma tecnologia padronizada e mundialmente utilizada permitindo que esta tenha um custo muito reduzido.

Rapidez - possui taxas de transmissão bastante elevadas. Até à data estas atingem 10, 100, 1000 e 10000 Mbit/s, e são, para a grande generalidade dos casos, mais que suficientes.

Escalabilidade - são redes que demonstram um desempenho idêntico, mesmo quando são alvo de grande crescimento. Este facto deve-se muito ao aparecimento de equipamentos, como os *hubs*, *switches*, *bridges* e *routers*, que aumentaram drasticamente a qualidade de serviço em redes alargadas.

Simplicidade - possui um sistema de interface e conexão, a nível físico e lógico, bastante simples, garantindo simplicidade e robustez do sistema.

Fiabilidade - possui um sistema de cablagem baseado em par-entrançado. Isto garante uma elevada imunidade ao ruído, diminuindo perdas, e tornando, consequentemente, o sistema bastante fiável.

Compatibilidade - garante compatibilidade entre produtos, nomeadamente quando estes suportam taxas de transmissão diferentes, o que torna o *upgrade* de sistemas bastante mais simples e barato.

2.3.2 Acesso ao meio

A tecnologia *Ethernet* usa um método de acesso ao meio, conhecido como *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD). Segundo este método, quando um nó da rede deseja transmitir verifica primeiro se a linha está livre. Desta propriedade (verificar a disponibilidade da linha) surge a designação de *Carrier Sense* e da propriedade de vários nós concorrerem pelo mesmo meio surge a designação de *Multiple Access*.

Se a linha se encontrar livre, o nó envia um pacote de informação. De seguida, o nó permanece à escuta, por modo a verificar se não ocorre nenhuma colisão. Uma colisão poderá ocorrer quando dois ou mais nós detectam que a linha está livre e enviam pacotes. Porém, estes ao propagarem-se pela linha acabam por colidir.

Caso não ocorra nenhuma colisão o nó sai do modo de transmissão. Em caso de colisão a transmissão ocorre até o tempo mínimo de duração de pacote ser atingido, por forma a garantir que todos os nós detectem a mesma. À propriedade de os nós conseguirem detectar colisões, dá-se a designação de *Collision Detection*.

Quando o meio é detectado como ocupado, os nós esperam até este estar livre. Quando fica livre, os nós esperam um tempo aleatório e tentam enviar de novo. Em caso de sucessivos fracassos no envio, o nó desistirá de enviar o pacote, ao fim de um número máximo de tentativas pré-estabelecido. A cada nova tentativa, a gama de valores que o tempo aleatório de espera pode tomar aumenta exponencialmente, segundo um algoritmo denominado de *Truncated Exponential Back-Off* [6, 12].

2.3.3 *Ethernet* em tempo-real

As redes do tipo *Ethernet*, apesar de muito populares, não são frequentemente usadas em sistemas com requisitos de tempo-real. Isto deve-se ao facto de possuirem indeterminismo temporal no envio dos pacotes. Este problema reside, fundamentalmente, no modo como é feito o acesso ao meio.

Para evitar o elevado crescimento de tráfego e consequente aumento de colisões, com o crescimento das redes foram desenvolvidos equipamentos designados de *switches*. Apesar de estes conseguirem diminuir, com bastante eficácia, o número de colisões na rede, não são suficientes para dar garantias de tempo-real, dado que o encaminhamento

de pacotes nos mesmos é feito, na maioria das vezes, por filas ordenadas por ordem de chegada, que pode muitas vezes levar a incumprimento de *deadlines*.

Este mecanismo de transmissão causa, além de uma ordem de chegada de pacotes indeterminada, o já anteriormente referido indeterminismo temporal no envio dos pacotes. Por estas razões, *Ethernet* não é considerado um barramento de campo, sendo contudo, possível conferir à *Ethernet* as propriedades necessárias para o ser. Há soluções que visam este propósito como, por exemplo, criar uma camada a nível superior que sincronize e escalone o envio de pacotes. Exemplos disto são as tecnologias *Ethernet Powerlink* [13] e *FTT-Ethernet* [14].

No entanto, em alguns sistemas a controlar cuja dinâmica é lenta e cujos requisitos de segurança não são críticos, estas limitações podem muitas vezes ser desprezadas, e a tecnologia *Ethernet* padrão pode ser utilizada sem trazer grandes limitações.

2.4 Linguagem Java

Java é uma linguagem de programação multi-tarefa e orientada a objectos. Tem origem num projecto denominado de *Green Project*, da *Sun Microsystems*. O objectivo original do projecto era o de criar um sistema que demonstrasse a tendência de interligação entre computadores e os equipamentos electrodomésticos do dia-a-dia e não a criação de uma linguagem de programação. Foi então criado um sistema, denominado de *7 (*Star Seven*), possuindo uma interface gráficas, capaz de controlar múltiplos aparelhos e aplicações. Para este efeito James Gosling, um dos seus mentores, especificou uma linguagem que denominou de *Oak*. Foram feitas várias tentativas de criar aplicações baseadas nesta linguagem, direccionadas a controlar sistemas de vídeo por demanda, mas depressa se concluiu que as empresas de televisão por cabo ainda não estavam preparadas para esta tecnologia. No entanto a rápida evolução da *Internet* providenciou a estrutura interactiva procurada. Assim, a linguagem *Oak* foi adaptada e em Janeiro de 1995 foi lançada ao público sob o nome de Java. Após o seu aparecimento, foi a linguagem de programação que, até à data, mais depressa proliferou e foi aceite em todo o Mundo. Em Maio de 2007, a Sun tornou a grande maioria do software Java *open source*, sob a licença *GNU General Public License* (GPL), aliciando ainda mais a sua utilização e desenvolvimento. Actualmente, a linguagem Java continua em constante desenvolvimento e conta com milhões de colaboradores em todo o Mundo.

2.4.1 Princípio de funcionamento

Java, ao contrário da maioria das linguagens de programação, foi construído segundo o conceito *"Write Once, Run Anywhere"* (WORA). Isto permite que o programador desenvolva uma aplicação, que a compile uma única vez e a corra em qualquer lado. Este aspecto deve-se, em muito, ao facto da linguagem Java ter sido desenvolvida para ser corrida em múltiplos sistemas através da *Internet*. O código desenvolvido nesta linguagem não tem uma compilação directa para código nativo da máquina. Este é transformado numa interpretação intermédia designada por *bytecodes*. Ao contrário do que acontece com outras linguagens que têm processos de compilação estáticos, a compilação dinâmica que a linguagem Java implementa não tem em conta a arquitectura da máquina, sendo *bytecode* um código independente. No entanto, é imprescindível que a arquitectura seja levada em conta durante o processo. Este passo é tomado pelo interpretador de *bytecodes*, designado de *Java Virtual Machine* (JVM). Este sim, depende da arquitectura, ou seja, uma arquitectura é compatível com qualquer código Java se existir uma máquina virtual desenhada para si. Actualmente, existem muitas máquinas virtuais Java disponíveis, compatíveis com sistemas como Windows, Linux, MacOS e alguns baseados em microprocessadores. Isto garante que as aplicações desenvolvidas em Java são compatíveis com quase todos os sistemas existentes, como indica a Figura 2.4.

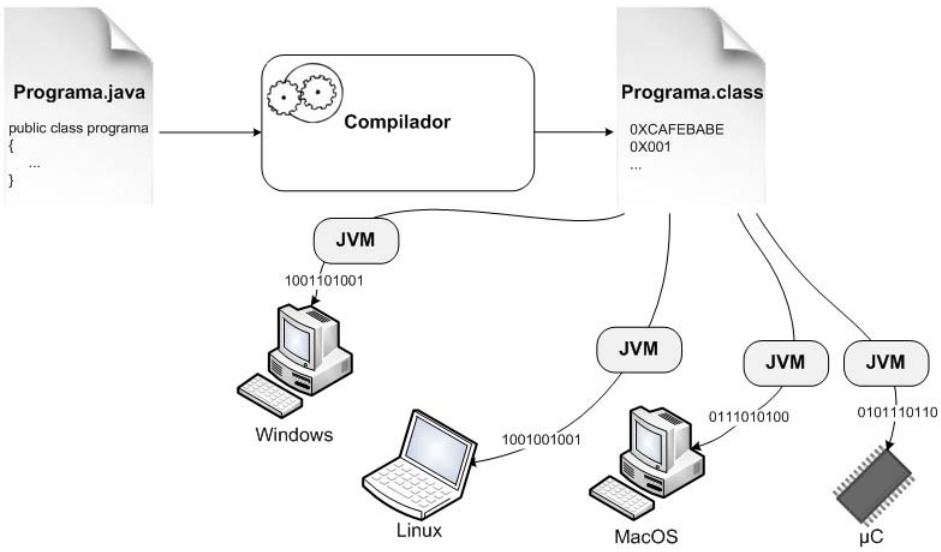


Figura 2.4: Compilação dinâmica em Java

2.4.2 Desempenho

Uma das questões que mais se põe em causa quando da ponderação de utilização da linguagem Java, é o seu desempenho. Isto prende-se com o facto de a sua compilação dinâmica implicar uma interpretação feita *online*, ao invés do que acontece em linguagens com compilação estática. Durante muito tempo estes receios foram bem fundamentados. No entanto, hoje em dia, com os avanços atingidos na implementação das máquinas virtuais, este efeito foi drasticamente minimizado, até ao ponto em que em muitos casos o compilador dinâmico chega a superar o estático. Pode-se afirmar que, nos dias que correm, a linguagem Java tem um desempenho já muito próximo de linguagens como C/C++. Isto deve-se, em muito, a mecanismos de optimização, como a compilação especulativa, que aproveita o tempo morto do processador para pré-compilar *bytecodes* para código nativo da máquina, ou a outros mecanismos que permitem que a máquina virtual vá "aprendendo" e optimizando o seu desempenho, entre outros. Além disto, o facto de a compilação ser feita *online* permite que o compilador tenha acesso a informação que só pode ser acedida durante a execução do programa, tal como o estado do sistema. Assim, a optimização pode ser feita com base neste, e não por um conjunto de parâmetros pré-estabelecidos com base no comportamento que, em média, este apresenta. Por todas estas razões existe uma forte tendência em acreditar que o futuro da programação está nas mãos de linguagens como o Java, uma vez que as restantes estão bastante evoluídas, mas pouco mais têm por onde melhorar. A linguagem Java, por outro lado, aparenta ainda ter muito por onde evoluir e parece prometer ultrapassar os seus competidores [15, 16].

Existe ainda, no entanto, uma desvantagem que a linguagem Java pode apresentar em termos de desempenho. A pré-compilação e carregamento da máquina virtual consomem um tempo considerável, o que torna o sistema relativamente lento a arrancar. No entanto, em aplicações como servidores, controladores, entre outros, que são idealmente inicializados uma única vez, esta desvantagem pode-se revelar insignificante, ainda para mais face às vantagens que o uso da linguagem poderá eventualmente trazer.

2.4.3 Segurança

Outra questão importante que se tornou bastante preocupante no uso da linguagem Java é a segurança. A forma como o código era passado para *bytecodes* permitia que fosse efectuado o processo designado de engenharia reversa. Ou seja, conseguia-se chegar através dos *bytecodes* ao código fonte original, algo que não é possível com a maioria das linguagens que passam o código directamente para código-máquina. Isto tornava a linguagem muitas vezes indesejável, principalmente para empresas de

desenvolvimento de *software*, que não queriam que o seu código fosse tornado público. Hoje em dia esta questão já não se põe. Existe já a possibilidade de reordenar e criptografar os *bytecodes*, tornando a engenharia reversa virtualmente impossível.

2.4.4 Principais vantagens

A linguagem Java possui várias vantagens, que são suficientes, na maioria dos casos, para justificar o seu uso. Algumas delas são [17]:

Linguagem orientada a objectos - sendo uma linguagem orientada a objectos, possui muitas vezes uma estrutura de *software* mais robusta, intuitiva e reutilizável. Isto facilita a fase de desenvolvimento e poupa recursos.

Multi-tarefa - possibilita a criação de programas com múltiplos fios de execução. Esta é uma propriedade muito importante dado que, na grande maioria das aplicações, existe a necessidade de simular processamento paralelo.

Portabilidade - a independência de plataforma torna o desenvolvimento mais simples, e torna também a aplicação utilizável em quase todos os tipos de sistemas. Isto torna, adicionalmente, o software desenvolvido nesta linguagem mais apetecível a potenciais compradores.

Utilização de recursos de rede - possui uma biblioteca muito completa para usos de protocolos TCP/IP, FTP e HTTP. Deste modo, facilita a criação de comunicações sobre redes.

Segurança - possibilita a execução de programas pela rede com restrições de execução, garantindo que estes só serão usados por quem tem autorização.

Sintaxe - possui uma sintaxe bastante simplificada e intuitiva, muito semelhante às de C/C++, que são linguagens bastantes usadas e bem documentadas, facilitando assim a aprendizagem em caso de uma possível transição. Foi, adicionalmente, fonte de inspiração para a linguagem C#, também muito conhecida e utilizada. De todas, esta será provavelmente a linguagem com a qual mais se assemelha.

Estrutura - possui uma estruturação modularizada e de fácil adaptação facilitando a especificação e modelização.

Facilidade de internacionalização - suporta nativamente caracteres *Unicode*, possibilitando assim a representação e manipulação consistente de qualquer sistema de escrita existente.

APIs - é distribuída com um vasto conjunto de bibliotecas, para os mais diversos propósitos. São exemplo as bibliotecas AWT ou Swing, que permitem a criação de

interfaces gráficas, a biblioteca IO que fornece métodos para entrada/saída de dados, ou a biblioteca NET para criação de aplicações distribuídas, entre uma enorme quantidade de outras [18].

Aplicação distribuída - criado originalmente para aplicações na *Internet*, possibilita a programação de aplicações distribuídas.

Gestão de memória - a desalocação de memória é feita de forma automática pelo *garbage collector*, acabando assim com erros de perda de memória que possam ser causados por falta de declaração formal pela parte do programador, indicando que a memória deverá ser desalocada. Contudo, como será referido mais à frente, este procedimento introduz um problema de indeterminismo temporal, dificultando a utilização de Java em tempo-real.

Carga dinâmica de código - as classes são armazenadas independentemente e podem ser carregadas apenas no instante de utilização. Isto torna o sistema mais leve, dado que este tem tendência a só ter em carga os módulos de que realmente necessita na altura.

***Open Source* -** é, actualmente, *software* com código de livre acesso, o que torna o seu uso ainda mais aliciante, quer do ponto de vista económico, quer do ponto de vista de desenvolvimento.

2.4.5 Java em sistemas de tempo-real

A utilização da linguagem Java em sistemas de tempo-real é algo que tem sido alvo de pouca aceitação. Isto deve-se às fortes limitações que possui ao nível do determinismo temporal. O processo de gestão de memória automático, conhecido como *garbage collector*, é uma tarefa com uma prioridade bastante elevada que provoca latências imprevisíveis. Além disso, o controlo de escalonamento é muitas vezes inadequado e o suporte para *timers* é bastante grosseiro. Não possui processamento de eventos assíncronos e a sincronização de *delays* é imprevisível

Actualmente já existe uma versão de Java para aplicações de tempo-real. É denominada de *Sun Java Real-Time System*, e resultou de uma especificação feita no sentido de fornecer à linguagem Java o rigor temporal necessário, ao controlo de sistemas com requisitos de tempo-real de segurança considerada crítica. Não deixa, contudo, de ser uma modificação à linguagem Java original, pelo que estas propriedades não se encontram presentes na maioria dos aplicativos existentes no mercado[19].

No entanto, em sistemas cuja dinâmica seja relativamente lenta e nos quais os requisitos temporais não sejam de segurança crítica, o facto do seu intervalo de amostragem

ser relativamente grande (da ordem de dezenas de segundos ou minutos) torna estas limitações poucos importantes, podendo até mesmo ser desprezadas. Perante este facto, a utilização da linguagem Java padrão, neste tipo de sistemas, poderá demonstrar-se de extrema utilidade, devido a todas as vantagens anteriormente referidas, principalmente aquando do desenvolvimento de sistemas de controlo distribuídos.

Mais detalhes sobre esta linguagem podem ser encontrados em [17].

2.5 Barramento 1-Wire

O barramento 1-Wire foi desenvolvido pela Dallas Semiconductor Corp sendo, também, uma marca registada da mesma. O seu nome deriva do facto de usar apenas um único fio para transmissão de dados. No entanto, o sistema necessita na realidade de dois fios, um para a referência e outro para alimentação e dados. O conceito presente no barramento 1-Wire é em muito semelhante ao presente em barramentos do tipo *Inter-Integrated Circuit*, vulgarmente conhecidos como barramentos I²C. Apresenta, no entanto, relativamente a estes, uma taxa de transmissão mais reduzida e um alcance superior. O barramento 1-Wire é do tipo *master-slave*, onde só pode existir um *master* em cada MicroLan. Esta é a designação que a Dallas atribui a uma rede de sensores 1-Wire com um *master* associado. Os *masters* podem ser do tipo PC ou microcontrolador, e acedem ao barramento através de porta série, paralela ou até USB. O propósito mais comum deste tipo de barramentos é o de comunicar com sensores de custo e tamanho reduzidos.

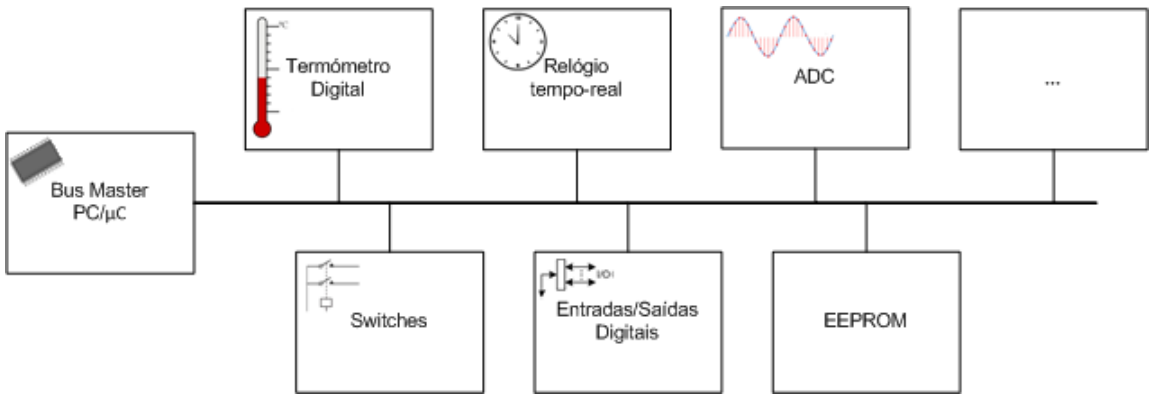


Figura 2.5: Barramento 1-Wire

A gama de sensores 1-Wire existente é muito alargada e cobre vários propósitos, tal como apresentado na Figura 2.5. Sensores de temperatura, monitores de baterias, sensores de corrente e tensão, memórias e *timers*, são apenas alguns exemplos. É

também frequente encontrarem-se sensores de utilidade mista, cobrindo mais que uma destas funcionalidades em simultâneo.

2.5.1 Principais vantagens

Os barramentos 1-Wire apresentam uma série de vantagens, tais como [20, 21, 22, 23]:

Custo - os sensores 1-Wire têm um custo muito reduzido.

Dimensão - apresentam um tamanho bastante reduzido o que torna fácil instalá-los em virtualmente qualquer lado.

Simplicidade - o barramento 1-Wire permite a implementação de uma rede alargada de sensores com bastante facilidade poupando muito em cablagem.

Escalabilidade - facilidade em adicionar um número elevado de sensores ao barramento sem comprometer o funcionamento do mesmo.

Alimentação - a alimentação dos sensores pode, na grande maioria das vezes, ser feita através do próprio barramento, mantendo, no entanto, alternativa de alimentação independente.

Alcance - uma rede 1-Wire bem projectada e com uma carga bastante elevada pode apresentar um alcance de algumas centenas de metros. Entenda-se por carga o número de sensores presentes no mesmo, e por alcance a maior distância de cabo entre o *master* e um *slave*.

Extensibilidade - possibilidade de adicionar *switchs* electrónicos, que vão comutando vários ramos da rede, possibilitando a expansão da rede mantendo a mesma carga, tal como apresentado na Figura 2.6.

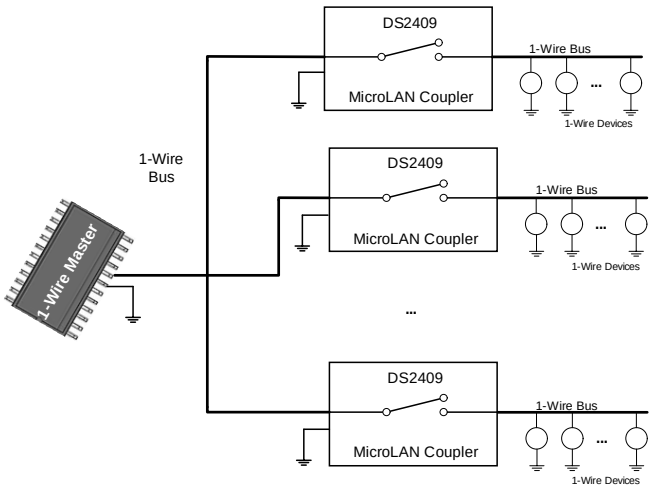


Figura 2.6: MicroLAN Coupler

Identificação inequívoca - cada sensor possui um número de série de fábrica único, composto por 64 *bits*. O *byte* menos significativo deste é usado para distinguir a família a que o dispositivo pertence. Isto garante unicidade, sendo vantajoso na gestão do barramento em si. O que conduz, adicionalmente, à sua grande popularidade em sistemas de gestão de acessos, onde os mesmos podem ser usados como chaves.

Método de busca - os sensores são procurados no seu espaço de endereçamento de 64 *bits* através do uso de uma árvore binária, o que permite encontrar até 75 dos mesmos por segundo.

2.5.2 1-Wire® API para Java™

Existe, actualmente, uma API 1-Wire para Java bastante completa. Esta permite o acesso a barramentos 1-Wire através de interfaces série, paralela, USB ou TCP/IP. Toma partido de todas as vantagens que esta linguagem apresenta, especialmente no sentido de tratar duma forma indistinta barramentos e sensores como objectos. Isto torna-se bastante vantajoso no desenvolvimento de sistemas garantindo simplicidade e robustez na programação. Torna, adicionalmente, fácil a criação de sistemas distribuídos a uma escala mais alargada, eventualmente global. Neste sentido, as limitações são virtualmente nulas. Suporta, de momento, quase toda a gama de sensores existentes, com raras excepções.

Qualquer PC, ou microcontrolador, com uma máquina virtual Java pode correr programas desenvolvidos com base nesta API. Um bom exemplo disto é o sistema baseado em microcontrolador *tiny Internet interface* da Maxim, conhecido como TINI, que inclui uma interface 1-Wire de raiz. Podem, hoje em dia, encontrar-se muitos sistemas baseados em 1-Wire e Java, tanto em plataformas PC, como microcontrolador, cobrindo áreas desde a meteorologia à domótica.

Não se deve, no entanto, deixar de referenciar a existência da *1-Wire Public Domain Kit*. Trata-se de uma API escrita em C, desenhada para ser portável em múltiplos PC's, com vários sistemas operativos, assim como em vários microcontroladores, com o intuito de controlar barramentos 1-Wire. No entanto, esta é uma API que não se aproxima, na generalidade, da simplicidade ou potencialidade da API em Java.

2.6 Norma SCPI

O *Standard Commands for Programmable Instruments* (SCPI) é uma norma que define um conjunto de comandos para o controlo de instrumentos de teste e medida em

sistemas de instrumentação. A norma surgiu com o propósito de fornecer uma linguagem comum a todos os instrumentos programáveis, dado que cada empresa possuía a sua própria linguagem, e estas tinham tendência a tornar-se cada vez mais complexas face ao desenvolvimento dos seus equipamentos. O que implicava que um cliente que comprasse produtos de empresas diferentes tivesse de se adaptar constantemente à linguagem do equipamento em causa.

As interfaces físicas a este tipo de equipamentos são geralmente GPIB, RS232, USB e VXIBUS. Isto em nada limita o uso de SCPI dado que esta norma não define a camada física de comunicação. O conjunto de comandos é formado por caracteres ASCII, que apresentam uma sintaxe legível pelo ser Humano, o que permite que seja extremamente fácil de comandar este tipo de equipamentos através do envio destas palavras, usando linguagens de programação como o C/C++, Java, entre outras [2].

2.6.1 Principais vantagens

A norma SCPI está, actualmente, presente na grande maioria dos equipamentos de instrumentação. Podemos encontrá-la em osciloscópios, multímetros digitais, entre outros, nomeadamente em marcas conceituadas, como Agilent, Berkeley Nucleonics e Tektronix. As vantagens que mais se destacam no seu uso são [2]:

Universalidade - sendo uma norma, tem a vantagem de ser comum à grande maioria dos equipamentos, poupando tempo de adaptação e garantindo a possibilidade de reutilização dos programas.

Simplicidade - usa uma sintaxe próxima ao Homem sendo, por isso, de uso intuitivo.

Independência de interface - usa símbolos ASCII, desprendendo-se de qualquer protocolo específico de interface com equipamentos. Assim sendo não limita o tipo de interfaces que podem ser usadas.

Acesso remoto - permite o acesso remoto aos equipamentos facilitando, adicionalmente, o tratamento dos dados provenientes dos mesmos em PC ou microcontrolador. Esta sua faceta demonstra-se de grande utilidade quando se pretende usar os dados para fins de controlo.

Capítulo 3

Utilização da linguagem Java em sistemas de controlo

A linguagem Java, tal como mencionado no capítulo 2, tem sido alvo de grande evolução. Actualmente a tendência é a de, cada vez mais, a utilizar na implementação de sistemas distribuídos. Apesar de menos frequente não são excepção a isto os sistemas de controlo distribuído. Mesmo tendo em conta todas as limitações associadas ao uso desta linguagem na implementação dos mesmos, tal como referido no capítulo 2, em muitos casos esta já é actualmente utilizada. Existem, adicionalmente, algumas adaptações à linguagem, igualmente referidas anteriormente, que visam proporcionar as características necessárias para alcançar este propósito.

O algoritmo de controlo é, normalmente, o processo da malha de controlo que possui maior carga de processamento. Esta carga não é muito elevada se nos referirmos a um simples controlador ON/OFF, um proporcional (P), um proporcional-integrador (PI), ou até um proporcional-integrador-derivativo (PID). No entanto, estes são tipos de controladores cujo desempenho é, muitas vezes, insuficiente face ao sistema em causa e aos objectivos propostos. Tendo isto em conta, é cada vez mais comum o uso de controladores com um maior grau de complexidade. São exemplo disto os controladores adaptativos ou os preditivos.

Dada a complexidade que muitos destes algoritmos de controlo apresentam torna-se, muitas vezes, vantajoso o uso de programas direccionados para a computação numérica. Este tipo de programas possuem a vantagem de incluir, normalmente, um conjunto de bibliotecas muito vasto que simplifica e optimiza a implementação do algoritmo de controlo. São exemplos deste tipo de programas o MATLAB, Scilab e Octave, entre outros.

A simplicidade de interface, característica da linguagem Java, torna simples a cons-

trução de mecanismos de interacção com este tipo de programas, aproveitando, assim, as suas propriedades e adicionando-lhes algumas potencialidades. Neste capítulo serão apresentadas algumas soluções possíveis na implementação de malhas de controlo baseadas na utilização conjunta de programas de computação numérica e linguagem Java.

3.1 Programas de computação numérica

Existem, actualmente, alguns programas direccionados à computação numérica que têm sofrido uma espantosa evolução. De entre estes, os mais conhecidos são o MATLAB, o Scilab e o Octave. Neste ponto será feita uma pequena introdução aos mesmos visando caracterizar, na generalidade, este tipo de programas, através destes três exemplos. É de salientar, no entanto, que o conceito não se aplica somente a estes. Servem apenas como guia, visto serem os mais utilizados.

3.1.1 MATLAB

MATLAB é o programa de computação numérica mais conhecido a nível mundial. Foi criado originalmente por Cleve Moler no fim dos anos 70. Mais tarde, em 1983, Little Jack juntou-se a Cleve Moler e Steve Bangert e reescreveram o programa MATLAB em C e fundaram a empresa MathWorks levando o MATLAB até ao mercado. O seu nome vem de *MATrix LABoratory* e, tal como este indica, o cálculo matricial é um dos seus pontos fortes. Este programa introduziu um conceito novo, pois permite que o utilizador se foque no trabalho em desenvolvimento, em vez de se focar em pormenores de programação. Isto tornou o desenvolvimento de projectos mais rápido e eficaz, relativamente ao atingido, até então, com linguagens de programação como C/C++, Fortran, entre outras. Esta sua propriedade tornou-se de tal forma famosa que, hoje em dia, é possível encontrar o MATLAB em múltiplas áreas, desde o controlo de sistemas, o processamento de sinal e o cálculo estatístico, entre muitas outras. Este programa tornou-se de tal forma conhecido e demonstrou tanto potencial, que outros programas surgiram com filosofias de funcionamento inspiradas no mesmo. São exemplos disto, os programas Scilab e Octave. Actualmente, o MATLAB encontra-se parcialmente escrito em Java e, por isso, corre a sua própria máquina virtual.

Apesar de não ser um programa do tipo *software open source*, revela, relativamente aos demais, muito mais potencial. É, de longe, o mais conhecido e utilizado e, consequentemente, o mais bem documentado. O seu conjunto de bibliotecas é muito vasto e completo, e o facto de possuir uma máquina virtual própria confere-lhe a propriedade

interessante de poder correr programas Java feitos pelo utilizador, o que se pode revelar de extrema utilidade.

3.1.1.1 Principais características

- Actualmente parcialmente escrito em Java, corre uma máquina virtual própria.
- Disponibiliza várias versões diferentes de venda ao público, englobando bibliotecas diferentes.
- Suporta cálculo matricial sendo esta, provavelmente, a sua funcionalidade com mais potencial.
- Existem distribuições de MATLAB para vários sistemas operativos nomeadamente Windows e Linux.
- Possui gestão de memória automática.
- A verificação de código é efectuada ao longo da escrita do mesmo (*dynamic typing*).
- Corre código Java e C.
- Dispõe de um conjunto muito vasto de bibliotecas.

3.1.1.2 Versão estudante

O MATLAB é vendido em várias versões. Cada versão é direccionada para um determinado âmbito. A versão estudante é uma versão muito utilizada no meio académico principalmente para ensino de álgebra, controlo e processamento de sinal. Esta versão inclui as seguintes bibliotecas:

- *Control System Toolbox* - *toolbox* de simulação, análise e parametrização de sistemas de controlo.
- *Signal Processing Toolbox* - *toolbox* para processamento de sinal.
- *Signal Processing Blockset* - conjunto de blocos Simulink para construção de sistemas de processamento de sinal.
- *Statistics Toolbox* - *toolbox* de cálculo estatístico.
- *Optimization Toolbox* - *toolbox* de optimização.

- *Symbolic Math Toolbox* - *toolbox* de cálculo simbólico.
- *Image Processing Toolbox* - *toolbox* de processamento de imagem.
- *Product Demos* - programas de exemplo.

Uma *toolbox* que não se encontra disponível na versão de estudante é a *Communications Toolbox*, que inclui funcionalidades como *sockets* TCP/IP e acesso a portas série.

3.1.2 Scilab

Scilab é um programa de computação numérica, desenvolvido pela *École Nationale des Ponts et Chaussées* (ENPC) e pelo *Institut National de Recherche en Informatique et Automatique* (INRIA). A criação deste programa foi profundamente influenciada pelo programa MATLAB. Em 2 de Janeiro de 1994 saiu a sua primeira versão como *software open source*. Desde então colaboradores de todo o Mundo criaram funcionalidades para o Scilab. Em 2003, o INRIA criou o Scilab Consortium, visando garantir uma contínua evolução e actualização do mesmo [24].

3.1.2.1 Principais características

- Trata-se de *software open source*, o que populariza o seu uso e propicia em muito o seu desenvolvimento por parte de colaboradores externos.
- Tem um elevado número de funções matemáticas.
- Suporta cálculo matricial sendo esta, provavelmente, a sua funcionalidade com mais potencial.
- Possibilita o desenvolvimento de funções próprias do utilizador, em linguagens como C/C++ e Fortran, entre outras.
- Tem uma sintaxe simples e semelhante a MATLAB. Integra, inclusivamente, um programa tradutor de código MATLAB para Scilab.
- Possui gestão de memória automática.
- A verificação de código é efectuada ao longo da escrita do mesmo (*dynamic typing*).

3.1.3 Octave

Octave é um aplicativo destinado ao cálculo numérico, igualmente inspirado no programa MATLAB. Foi, originalmente, criado por James B. Rawlings da Universidade de Wisconsin-Madison e por John G. Ekerdt da Universidade do Texas por volta de 1988. No entanto, a sua primeira versão só foi lançada a 3 de Janeiro de 1993. Deste então tem sido alvo de várias revisões e várias funcionalidades foram adicionadas por contribuidores externos.

3.1.3.1 Principais características

- Trata-se de *software open source*, o que populariza o seu uso e propicia em muito o seu desenvolvimento por parte de colaboradores externos.
- Dispõe de um elevado número de funções matemáticas.
- Suporta cálculo matricial sendo esta, provavelmente, a sua funcionalidade com mais potencial.
- Possibilita o desenvolvimento de funções próprias do utilizador, em linguagens como C/C++ e Fortran, entre outras.
- Tem uma sintaxe simples e semelhante ao MATLAB.
- Possui gestão de memória automática.
- A verificação de código é efectuada ao longo da escrita do mesmo (*dynamic typing*).

3.2 Controladores baseados em programas de computação numérica

Tal como referido anteriormente a utilização de programas de computação numérica, em malhas de controlo é algo que se pode revelar muito vezes vantajoso. No restante capítulo serão apresentados alguns aspectos sobre este conceito e algumas soluções possíveis para a sua implementação.

A implementação referida segue a estrutura apresentada na Figura 3.1.

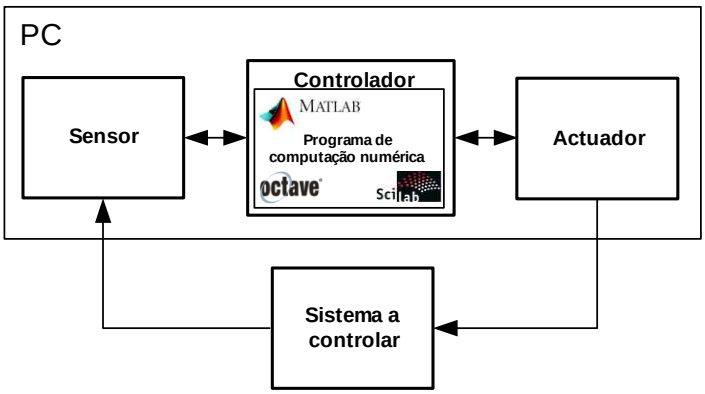


Figura 3.1: Controlador baseado em programa de cálculo numérico (monolítico)

Como pode ser observado na Figura 3.1 o programa de cálculo numérico é um elemento muito importante do sistema. É inteiramente sobre este que o controlador é implementado. Ou seja, este é responsável por decidir como actuar no sistema face às informações que recebe dos sensores. Assim sendo é muito importante garantir o seu correcto funcionamento e coordenação com o restante sistema, por forma a garantir que o sistema a controlar apresentará o comportamento estipulado.

As vantagens que este tipo de solução apresenta são muitas. Por outro lado, estas podem muitas vezes ser acrescidas se a solução for distribuída tal como apresentado na Figura 3.2.

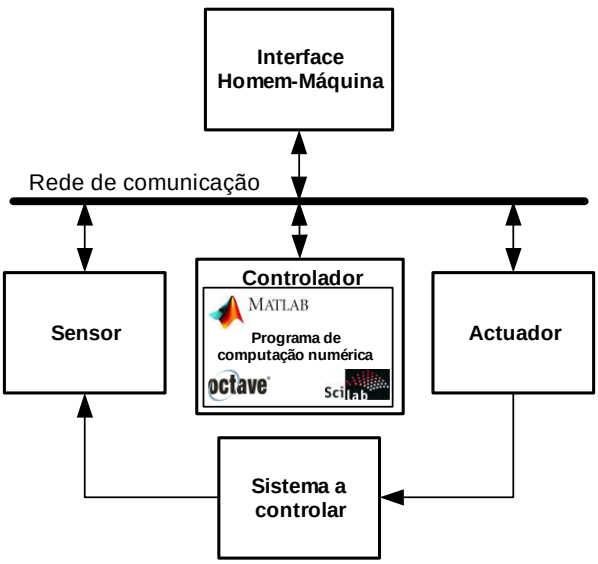


Figura 3.2: Controlador baseado em programa de cálculo numérico (distribuído)

Este tipo de solução visa a implementação de uma malha de controlo distribuída cujo controlador é, igualmente, implementado inteiramente sobre um programa de computação numérica. Esta estrutura apresenta relativamente à anterior todas as vanta-

gens que as soluções distribuídas apresentam, relativamente às monolíticas, mencionadas no capítulo 2. Na sua implementação devem, igualmente, ser tidos em conta os mesmos aspectos que na sua versão monolítica.

3.2.1 Comunicação

Uma questão, muito importante, que surge durante o processo de implementação de uma malha de controlo, usando um controlador baseado num programa de computação numérica, é como fazer a comunicação com o exterior. Esta questão prende-se, principalmente, às propriedades do programa em causa e do tipo de comunicação que melhor se enquadra no sistema a implementar. Destes dois pontos rapidamente se conclui que muito dificilmente se encontra uma solução que se possa considerar ideal.

Várias hipóteses surgem na interface do controlador com os sensores e os actuadores. Uma solução possível, e bastante popular, é o acesso por porta série. Muitos são os sistemas de instrumentação com suporte a esta tecnologia. São exemplos disto os equipamentos de instrumentação com suporte à norma SCPI e adaptadores para acesso a barramentos do tipo 1-Wire, entre outros. A sua popularidade deve-se, maioritariamente, à sua simplicidade de implementação e robustez. A maioria dos programas de computação numérica possuem rotinas de acesso a portas série, tornando assim viável a sua utilização em conjunto com uma alargada gama de equipamentos. Porém, muitas vezes este tipo de rotinas não se encontram disponíveis em todos os programas de computação numérica e nalguns não são gratuitas. Existem APIs que permitem o acesso a portas série usando linguagem Java. Torna-se então, em alguns casos, uma alternativa viável, servindo de intermediário no acesso a este tipo de interfaces. Esta solução tem a vantagem da portabilidade da linguagem Java e do facto de não representar custos acrescidos.

Outro tipo de comunicações muito usual são os *sockets*. Estes operam, tipicamente, sobre sistemas distribuídos com redes de comunicação do tipo *Ethernet*. No entanto, não deve ser esquecido que estes não estão disponíveis em todos os programas de computação numérica, e nalguns, onde estes estão presentes, esta funcionalidade é paga, como é o caso da *Communications Toolbox* do MATLAB. Adicionalmente, criar uma solução baseada em *sockets* nativos dos programas, implica uma implementação diferente consoante o programa em causa. A linguagem Java suporta, nativamente, a utilização de *sockets*, sendo a implementação de programas baseados nos mesmos de fácil desenvolvimento e elevada simplicidade. Deve, também, ser tido em conta, mais uma vez, que esta escolha não acarreta quaisquer custos adicionais, o que a torna ainda mais vantajosa.

Apesar das soluções expostas, a grande maioria dos programas de computação numérica não possibilitam a execução de código Java, factor que deve ser levado em conta. No entanto, não deixa de ser possível a sua utilização, na grande maioria das vezes, na interacção com programas de computação numérica, fornecendo este tipo de interfaces de comunicação, de uma forma directa ou indirecta.

Tendo em conta estes aspectos, serão seguidamente apresentadas algumas propostas de implementação de malhas de controlo, com controladores baseados em programas de computação numérica. As diferentes soluções apresentadas terão, muitas vezes, em conta o uso de *sockets* TCP e acesso a portas série em Java, no entanto vão diferir em alguns aspectos estruturais, consoante o leque de programas a que se destinem.

3.2.1.1 TCP vs UDP

A implementação de comunicações baseadas em *sockets* pode ser feita usando os protocolos TCP ou UDP. Para as implementações em causa foi dada preferência ao protocolo TCP em detrimento do UDP, pelas razões que se seguem:

- Maior simplicidade de implementação.
- Garantia de ordem correcta de chegada de pacotes.
- Garantia de recepção de pacotes.

De notar que, nesta escolha, foi tida em conta a carga adicional de tráfego que o TCP apresenta, relativamente ao UDP. No entanto, sob os pressupostos apresentados no capítulo 1, este efeito foi considerado desprezável.

3.2.2 Marca temporal

Em todas as soluções de implementação que serão seguidamente apresentadas, a marca temporal encontra-se sempre presente no elemento controlador, ou seja, esta será gerada pelo programa de computação numérica, pelo que é este que toma a iniciativa de estabelecer todas as comunicações.

3.3 Solução A: Código Java em MATLAB

Esta solução de implementação baseia-se na interpretação de código Java a partir da aplicação de computação numérica. É, portanto, exclusivamente direccionada para MATLAB, dado que este é o único que possui a sua própria máquina virtual.

Para este efeito o código Java pode ser guardado numa M-file e executado em qualquer altura. Esta implementação tem a vantagem de poder aproveitar as potencialidades e facilidades que o MATLAB apresenta na área de computação numérica e de apresentação gráfica, ao mesmo tempo que utiliza funcionalidades da linguagem Java, tais como a instanciação de objectos, criação de *sockets* e acesso a portas série, entre outras. A Figura 3.3 apresenta a estrutura da solução proposta.

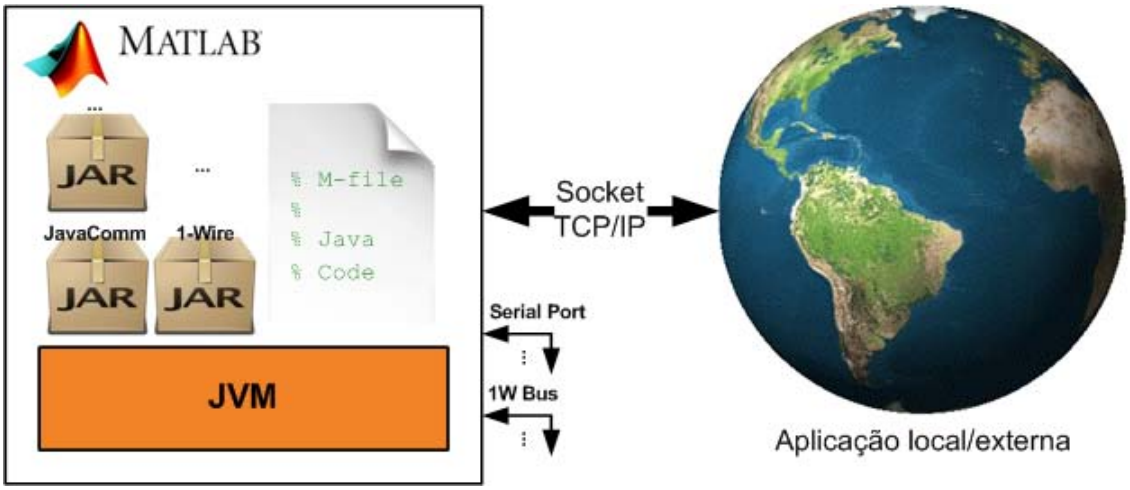


Figura 3.3: Execução de código Java em MATLAB

Este tipo de estrutura resolve a questão anteriormente colocada da comunicação, possibilitando que o controlador residente no MATLAB comunique com os outros elementos do sistema. Estes elementos poderão potencialmente ser sensores, actuadores ou até interfaces Homem-Máquina. A utilização de *sockets* permite, adicionalmente, que estes elementos se localizem virtualmente em qualquer parte do Mundo. No entanto, estes elementos estão muitas vezes igualmente presentes na mesma entidade computacional que o controlador, podendo o acesso ser feito para o endereço local (*localhost*), no caso do uso de *sockets*.

No acesso a todas as aplicações exteriores ao MATLAB (locais e externas), este é quem toma a iniciativa de estabelecer a marca temporal e as comunicações, tal como anteriormente referido. Ou seja, é este que gere o acesso a todo o tipo de comunicações, desde o acesso a portas série ou à utilização de *sockets*. No uso de *sockets*, adopta-se a filosofia das aplicações, exteriores ao MATLAB, correrem um servidor TCP, enquanto que o MATLAB corre um cliente. Isto permite que estas estejam sempre disponíveis e acessíveis ao MATLAB, enquanto que este pode correr um simples cliente, que só acede a estas quando necessário.

Por outro lado, poderão ainda ser utilizadas, a partir do MATLAB, API's como a Javacomm da Sun, que permite o acesso a portas série e paralela, ou a 1-Wire

API da Maxim, que permite o acesso a barramentos 1-Wire, entre outras. Nesta perspectiva, estes acessos seriam, ao contrário do permitido pelo uso de *sockets*, todos feitos localmente. Para a sua utilização é necessária a sua inclusão prévia no *classpath* do MATLAB de modo a que este os carregue no arranque e conheça, desta forma, as suas classes. Este processo é descrito no apêndice B.

Esta versatilidade, torna fácil a estruturação de um sistema à medida dos objectivos propostos. Facilmente se pode criar um sistema com acessos locais e remotos, e com capacidade de interface à grande maioria dos sistemas de instrumentação existentes.

Algorithm 3.1 Listagem de sensores presentes num barramento 1-Wire

```
clear all
close all

%Import das classes da OneWireAPI necessárias
import com.dalsemi.onewire.OneWireAccessProvider;
import com.dalsemi.onewire.adapter.DSPortAdapter;
import com.dalsemi.onewire.container.OneWireContainer;
import java.util.Enumeration;

%Adaptador ligado ao PC
adapter='DS9490';

%Porta de interface com o adaptador
port='USB1';

%Objecto de acesso ao barramento
owBus=OneWireAccessProvider.getAdapter(adapter, port);

%Garantia de acesso exclusivo ao barramento
owBus.beginExclusive(true);

%Ajuste do filtro de procura para todas as famílias de sensores
owBus.setSearchAllDevices();

%Busca de todos os sensores
containers=owBus.getAllDeviceContainers();

%Listagem de todos os nomes e chaves dos sensores presentes no barramento
while(containers.hasMoreElements())
owc=container.nextElement();
disp([owc.getName() owc.getAddress()]);
end

%Fim de exclusividade de acesso ao barramento
owBus.endExclusive();

%Libertação da porta de interface ao barramento
owBus.freePort();
```

3.3.1 Exemplo de utilização

Neste exemplo é apresentado um trecho de código que quando executado no MATLAB permite o acesso a um barramento do tipo 1-Wire e listagem de todos os sensores presentes no mesmo. Trata-se de um exemplo de código Java executado em MATLAB que utiliza conjuntamente a 1-Wire API e a JavaComm, previamente incluídas no seu *classpath*, para o acesso a barramentos do tipo 1-Wire. Este exemplo encontra-se listado no Algoritmo 3.1.

3.4 Solução B: API Java em MATLAB

Esta solução é, em muitos aspectos, semelhante à apresentada anteriormente. É igualmente direccionada ao uso em MATLAB fazendo uso da máquina virtual que o mesmo possui. No entanto, difere nalguns aspectos estruturais. Em vez de o código Java ser executado directamente "em cima" de uma M-file, é criada uma API própria do utilizador, onde este inclui todos as classes de que necessita. Esta API é guardada num ficheiro *Java Archive* (JAR), e pode ser incluída no caminho de classes do MATLAB. A implementação proposta segue a estrutura presente na Figura 3.4.

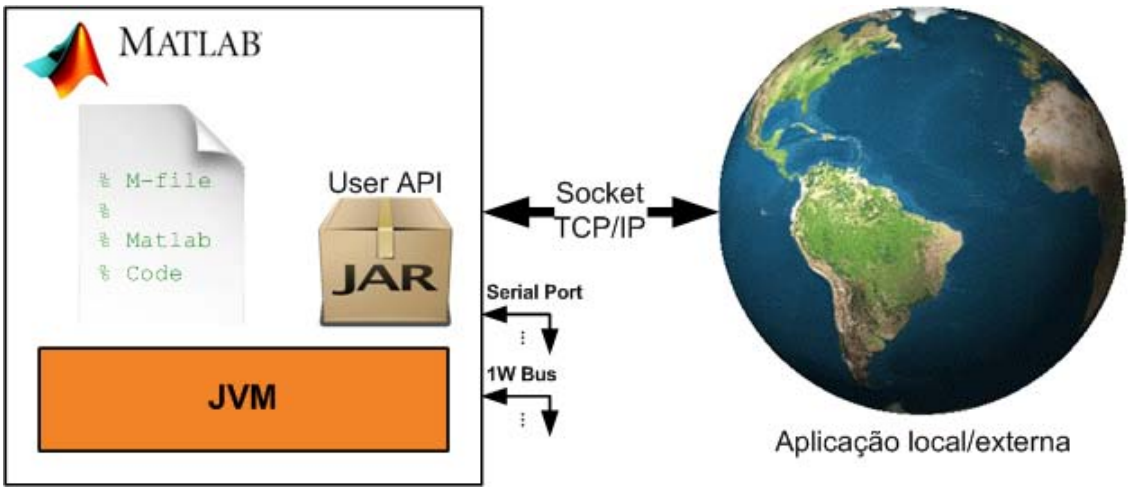


Figura 3.4: API Java em MATLAB

Esta estrutura visa, igualmente, a possibilidade de se incluírem outras API's, proporcionando acesso a uma série de interfaces locais, e o uso de *sockets* é, também, possível na comunicação com aplicações exteriores ao MATLAB. As comunicações por *sockets* seguem a mesma filosofia de cliente-servidor, dado que o MATLAB continua a ser o elemento activo do sistema.

Algorithm 3.2 Criação de um cliente TCP

```
%Criação do objecto cliente indicando um ip e um porto
client=ClientTCP('192.168.9.160',5000);

%Abertura do canal de comunicações
client.create();

%Escrita para o socket destino
client.write('Teste de cliente TCP');

%Leitura de resposta
response=client.read();

%Resultado
disp(['Resposta:' response]);

%Fecho do canal de comunicações
client.destroy();
```

Esta solução possui, relativamente à anterior, a vantagem de criar uma maior abstracção. Além disso, permite um melhor controlo do código em linguagem Java, dado que quando este é corrido em MATLAB necessita de alguns pequenos ajustes associados a propriedades inerentes ao mesmo, tal como, por exemplo, a declaração automática de variáveis, que faz com que uma instância de uma classe não tenha de ser explicitamente declarada, enquanto que em código Java original tem.

3.4.1 Exemplo de utilização

Neste exemplo de utilização é usado um *Java Archive* criado no âmbito desta dissertação com o nome javaOnMatlab.jar (esta API é apresentada no apêndice C), mais especificamente uma das classes que inclui chamada ClientTCP que, tal como o nome indica, permite criar clientes usando o protocolo TCP. Trata-se de um exemplo simples em que é criado um cliente que acede a um servidor, envia uma mensagem e recebe uma resposta que posteriormente imprime na consola. Este exemplo encontra-se listado no Algoritmo 3.2.

3.5 Solução C: *Gateway* remota para acesso a periféricos

As duas soluções anteriormente apresentadas permitem o acesso a um série de instrumentação local, assim como comunicação remota via *sockets*. No entanto, estas podem ser facilmente estendidas por forma a tomarem partido da existência de equipa-

mentos de instrumentação em locais remotos, fazendo-se para isso valer da já referida comunicação por *sockets* existente em ambas.

Para este efeito, considera-se a existência de uma *gateway* que corre um servidor acessível por *socket*. Assim, o cliente existente em ambas as soluções anteriores poderá aceder a este efectuando pedidos de leitura ou actuação. Isto permite que o controlador tenha a capacidade de aceder a sensores ou actuadores existentes fora da entidade computacional onde reside. No entanto, não deve deixar de ser referido, mais uma vez, que este acesso pode, igualmente, ser feito a uma *gateway* que resida na mesma entidade computacional, sendo assim o acesso feito por *localhost*. Esta solução, segue a estrutura presente na Figura 3.5.

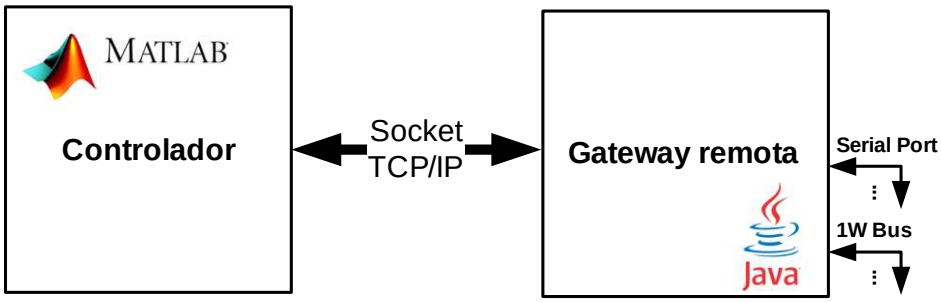


Figura 3.5: *Gateway* remota para acesso a periféricos

Facilmente se poderá com esta solução criar o acesso pela parte do controlador a interfaces remotos, tais como portas série, barramentos 1-Wire, equipamentos de instrumentação variados, entre outros, que estejam virtualizados na rede pela *gateway* que corre na entidade computacional onde estes residem.

3.5.1 *Gateway* remota

A gateway remota anteriormente referida é totalmente implementada em linguagem Java. Esta tem a capacidade de se tornar acessível remotamente via um servidor TCP, que permite o acesso a interfaces presentes na entidade computacional onde reside. A *gateway* funciona segundo uma filosofia de comunicação do tipo comando-resposta, ou seja, o servidor espera que um cliente lhe envie um comando, processa-o e envia uma resposta, passando novamente a um estado de espera, tal como demonstrado na Figura 3.6.

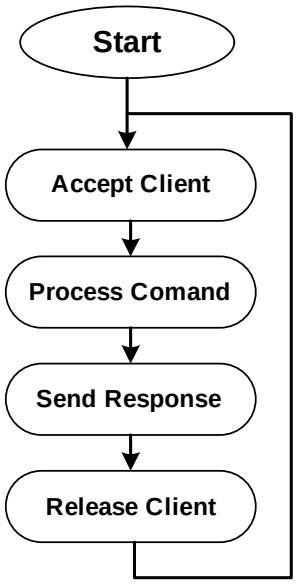


Figura 3.6: Diagrama de estados *Gateway*

O tipo de pedidos aceite pela gateway são comandos ASCII que assumem uma hierarquia em ramos separados por ponto e vírgula. A gama de comandos disponíveis, de momento, é apresentada na Figura 3.7.

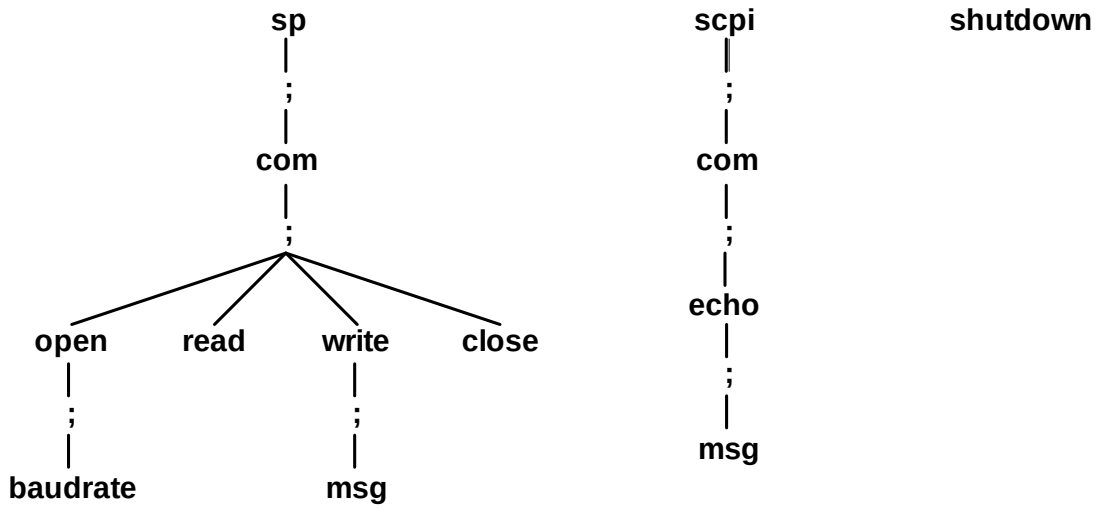


Figura 3.7: Gama de comandos *gateway* remota

Os comandos apresentados permitem o acesso remoto a portas série e equipamentos que utilizem a norma SCPI. No entanto, a estruturação de código da *gateway* permite com grande facilidade a adição de novas funcionalidades. A título de exemplo, se for pretendida a abertura da porta COM1 a uma baudrate de 9600 deverá ser enviado o comando *sp;COM1;open;9600*.

3.5.2 Exemplo de utilização

Neste exemplo de utilização é feita uma leitura remota de temperatura de um termopar utilizando um datalogger HP34970A. É feita uma utilização conjunta de MATLAB, da API Java do utilizador (javaOnMatlab.jar) anteriormente referida e da *gateway* remota. Para o efeito o trecho de código presente no Algoritmo 3.3 é executado no MATLAB.

Algorithm 3.3 Leitura remota de temperatura de um termopar

```
%Porta COM remota onde o datalogger está ligado
comDL='/dev/ttyUSB0';

%Cliente para acesso à gateway
gateway=ClientTCP('192.168.9.160',5000);

%Abertura da porta série
checkPortsCom=sendGateway(gateway,['sp;' comDL ';open;9600']);

%Verificação de sucesso
if strcmp(checkPortsCom,[comDL 'does not exist.'])==1
    disp([comDL ' does not exists.' ' Program will be aborted.']);
return;
end

%Leitura da temperatura do termopar usando a norma SCPI
response=str2num(sendGateway(gateway,['scpi;' comDL ';true;MEAS:TEMP? TC,K,
(@105)']));

%Apresentação de resultado
disp(['Temperatura: ' response 'ºC']);
```

A função *sendGateway* presente no trecho de código anteriormente apresentado é uma função simples desenvolvida em MATLAB que permite o envio de comandos para a *gateway* remota e recebe as respostas quando for caso disso. Esta função é constituída pelo código apresentado no Algoritmo 3.4.

Algorithm 3.4 Função *sendGateway*

```
function echo=sendGateway(gateway,msg)
gateway.create();
gateway.write(msg);
if(length(strfind(msg,'false'))==0)
    echo=gateway.read();
end
gateway.destroy();
```

3.6 Solução D: *Gateway* intermédia baseada em ficheiros

Esta implementação surge, igualmente, como uma possível solução para contornar o problema de comunicação com o exterior em outros programas que não o MATLAB. Contudo não deixa de ser, igualmente, aplicável ao mesmo. Ou seja, viabiliza, de igual modo, uma solução baseada em programas como, por exemplo, o Scilab ou o Octave. A estrutura que toma deve-se ao facto de, à excepção do MATLAB, estes programas de computação numérica não permitirem, à data, a execução de código Java, tal como anteriormente mencionado. Esta solução usa uma aplicação intermédia desenvolvida em Java, totalmente independente do programa de computação numérica em causa.

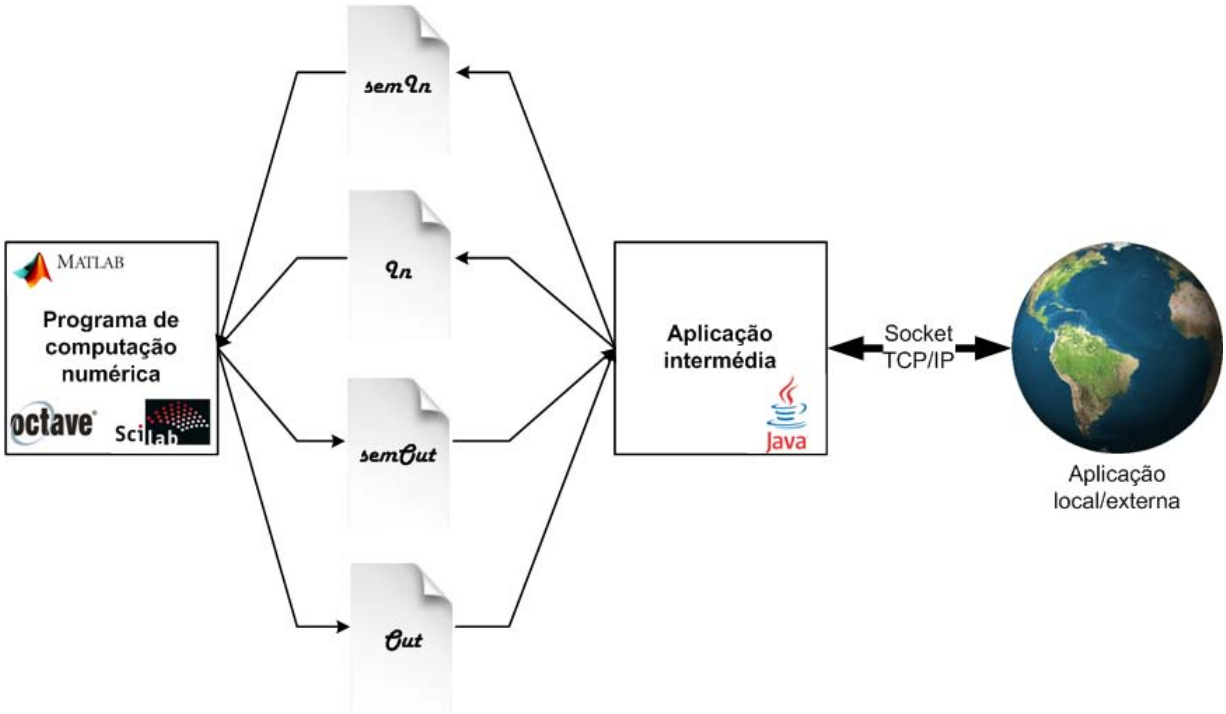


Figura 3.8: Aplicação intermédia baseada em ficheiros

A comunicação do programa de computação numérica com o exterior segue a mesma filosofia que a apresentada nos casos anteriores. Ou seja, o programa em causa é que toma a iniciativa de comunicar com o exterior, definindo assim a marca temporal do sistema de controlo. Porém, neste caso existe a necessidade de retirar a comunicação por *sockets* de dentro do programa, dado que este é o factor limitativo. Assim sendo, é feita uma comunicação baseada em ficheiros de texto entre o programa de computação numérica e a aplicação intermédia. É nesta aplicação que se encontra um cliente TCP que faz o encaminhamento da informação vinda de, e para, o programa de computação

numérica. Poderá, portanto, ser visto como uma *gateway* ficheiro-*socket*. Este mecanismo baseia-se na gestão de quatro ficheiros, dois que servem de semáforos e outros dois para transferência de dados. Existe um semáforo e um contentor de dados para cada direcção de comunicação. Os semáforos têm a função de indicar o estado de cada contentor de dados. Quando o programa de computação numérica quer comunicar com o exterior, escreve no contentor de dados e, de seguida, indica-o no semáforo respectivo. A aplicação intermédia deverá estar constantemente à escuta de dados vindos do programa de computação numérica e, quando estes chegam, lê-os do contentor e envia-os pelo *socket*, actualizando o valor do respectivo semáforo. Na comunicação contrária o procedimento é o oposto. Existe adicionalmente um ficheiro para configuração em *runtime* do ip e porto de acesso. A estrutura básica desta solução é apresentada na Figura 3.8.

Esta solução é bastante abrangente, principalmente porque a grande maioria dos programas de computação numérica suportam o acesso a ficheiros de texto. Assim sendo, torna-se muitas vezes uma solução bastante viável e simples de implementar.

3.6.1 *Gateway* intermédia

A estrutura anteriormente apresentada utiliza um sistema de semáforos por forma a garantir exclusividade de acesso e, consequentemente, a integridade da informação a transmitir.

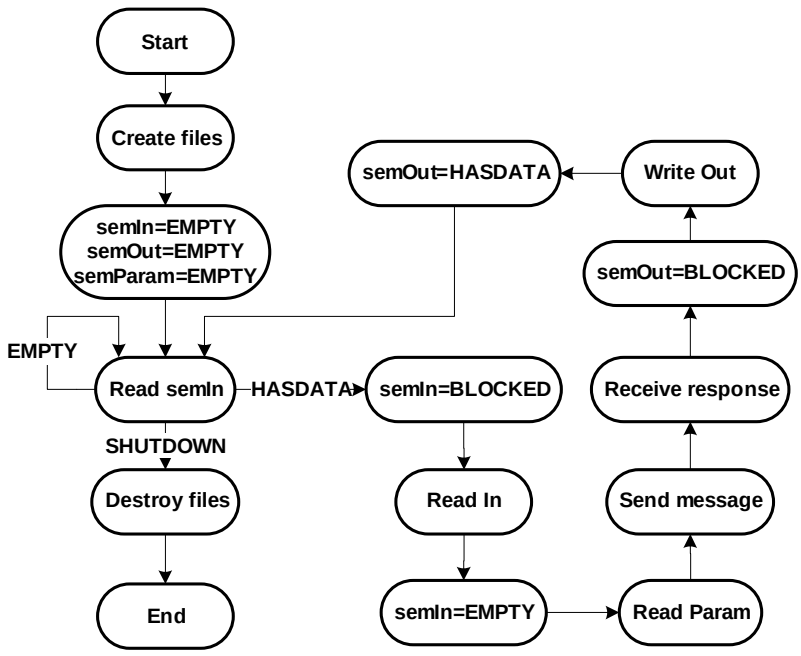


Figura 3.9: Fluxograma *gateway* intermédia

O modo de funcionamento da *gateway* pode ser representado pelo fluxograma presente na Figura 3.9.

3.6.2 Exemplo de utilização

Neste exemplo de utilização é feito o acesso remoto a uma porta série. Para o efeito é utilizado o programa de computação numérica Scilab e as *gateways* intermédia e remota, anteriormente mencionadas. O trecho de código necessário é apresentado no Algoritmo 3.5.

Algorithm 3.5 Escrita remota em porta série

```
//Carregamento das funções necessárias
funcprot(0)
getf("read.sci");
getf("write.sci");
getf("config.sci");

//Porta série remota a aceder
com='/dev/ttyUSB0';

//Configuração de ip e porto a aceder
config('192.168.9.160:5000');

//Abertura da porta série
write('sp;'+com+';open;9600');

//Escrita para a porta série remota
write('sp;'+com+';write;GatewayTeste');
```

As funções *write*, *read* e *config* são funções desenvolvidas em Scilab, no âmbito desta dissertação, que fazem a gestão dos ficheiros de texto do lado do Scilab de uma forma semelhante à da *gateway* intermédia.

3.7 Resumo

Neste capítulo foram abordadas algumas soluções possíveis para a implementação de malhas de controlo baseadas em programas de computação numérica e linguagem Java. Nestas soluções foram tidas em conta soluções monolíticas e distribuídas e foram referidas algumas das suas principais características, vantagens e diferenças. No capítulo seguinte, serão abordados alguns casos práticos cuja implementação é baseada em algumas destas soluções.

Capítulo 4

Exemplos de aplicação

Neste capítulo são apresentados alguns exemplos de aplicação prática dos conceitos expostos no capítulo anterior. É apresentada a sua estrutura e filosofia de funcionamento, bem como resultados práticos da sua aplicação na aquisição e controlo em alguns sistemas.

Em primeiro lugar é apresentada uma solução de controlo distribuído, que visa controlar um processo térmico usando MATLAB e linguagem Java. Em segundo lugar, é apresentada uma solução para controlo do mesmo processo usando como alternativa o Scilab. Por último, é apresentado um sistema distribuído de aquisição baseado na utilização conjunta de barramentos do tipo 1-Wire, Java e MATLAB.

4.1 Controlo de processo térmico baseado em MATLAB e Java

Este exemplo de aplicação visa o controlo da temperatura do fluxo de água de saída do módulo PCT 13 da Armfield, apresentado no Apêndice A. Para o efeito, a filosofia de comunicação segue a estrutura presente na secção 3.5. O controlador é, portanto, implementado totalmente em MATLAB e as comunicações para aquisição de sensores e actuação são efectuadas remotamente via *sockets*. O acesso a *sockets* é totalmente implementado numa classe em linguagem Java, desenvolvida para o efeito, estando incluído num *Java Archive* que é adicionado ao *classpath* do MATLAB, sendo assim possível invocá-los em código a partir do mesmo. A interface com sensores e actuadores é feita remotamente no acesso a uma Mini-ITX D945GCLF da Intel que corre a *gateway* remota descrita na secção 3.5.1.

4.1.1 Mini-ITX D945GCLF

A Mini-ITX D945GCLF é uma placa produzida pela Intel cujas características a tornam uma boa escolha para uso em sistemas remotos, com localização geográfica de difícil acesso. Possui dimensões reduzidas (171.45 milímetros por 171.45 milímetros) e vários tipos de interfaces tais como USB 2.0, RS232, porta paralela, entre outros. É, portanto, um tipo de PC com dimensões e funcionalidades reduzidas, características muito comuns em sistemas embutidos. Esta placa é apresentada na Figura 4.1.

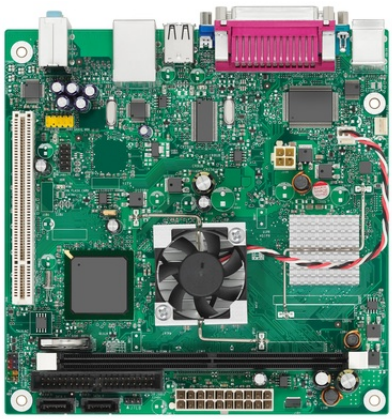


Figura 4.1: Mini-ITX D945GCLF

Para os testes a efectuar no âmbito desta dissertação foi instalado, na Mini-ITX, o sistema operativo Linux Ubuntu 9.04, juntamente com um servidor SSH, para que esta pudesse ser acedida e configurada remotamente.

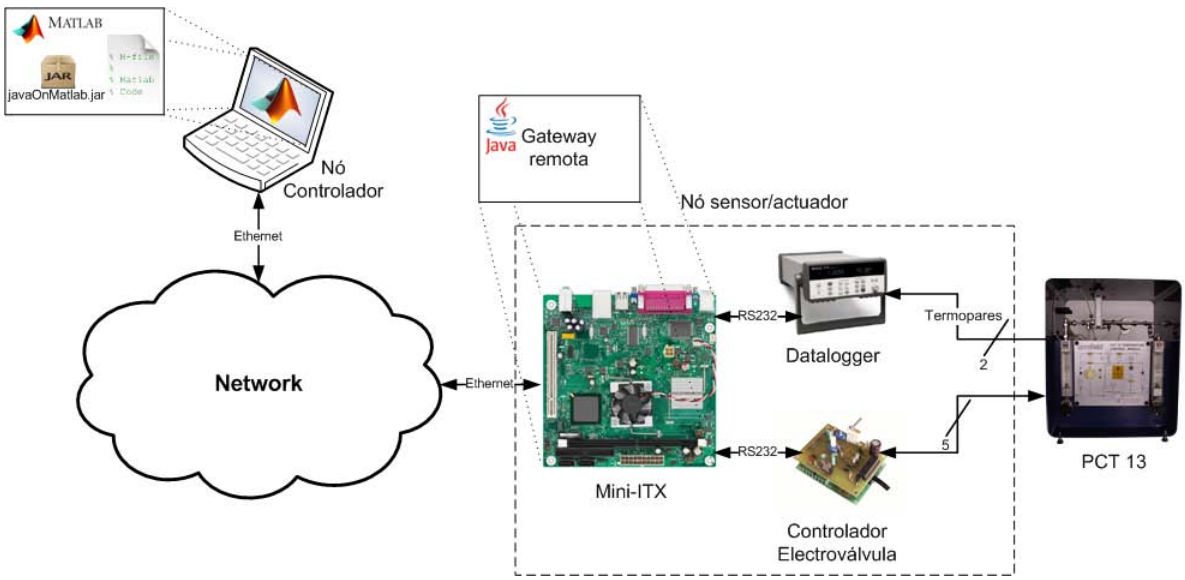


Figura 4.2: Exemplo de aplicação 1

4.1.2 Estrutura do sistema

O sistema de controlo distribuído deste exemplo de aplicação segue a estrutura apresentada na Figura 4.2. Como se pode observar este exemplo utiliza uma rede de comunicação do tipo Ethernet. O nó controlador é composto por um PC com grande capacidade de cálculo que corre o programa MATLAB. O nó sensor/actuador é composto pelo PC Mini-ITX, um *datalogger* HP34970A e uma placa de controlo de electroválvula, desenvolvida para o efeito no âmbito desta dissertação. A interacção da Mini-ITX com o *datalogger* e com controlador de electroválvula é feita via RS232. O *datalogger* permite a leitura dos termopares presentes no módulo PCT 13 de uma forma simplificada, usando para o efeito a norma SCPI. O controlador da electroválvula permite um controlo simplificado da posição de abertura da electroválvula, bastando o envio de um *setpoint* via RS232. A *gateway* anteriormente referida está presente na Mini-ITX e permite a gestão destes acessos.

4.1.3 Algoritmo de controlo

O sistema de controlo deste exemplo de aplicação utiliza um algoritmo de controlo adaptativo por posicionamento de pólos com tempo morto fraccionário descrito em [10, 25, 26]. A referência a tempo morto fraccionário deve-se ao facto de a caracterização do módulo PCT 13 presente no apêndice A ter revelado a existência de um tempo morto fraccionário no subsistema electroválvula de aproximadamente 4 segundos. O algoritmo de controlo utilizado possui um estimador de 1ª ordem que gera um modelo matemático do sistema em tempo-real. O modelo matemático do sistema é apresentado na expressão 4.1.

$$y(k) = \hat{\theta}_1 y(k-1) + \hat{\theta}_2 u(k-1) + \hat{\theta}_3 u(k-2) \tag{4.1}$$

$\hat{\theta}_1$, $\hat{\theta}_2$ e $\hat{\theta}_3$ são os parâmetros obtidos do estimador de tempo-real.

4.1.4 Condições de teste

Para o teste de funcionamento do sistema de controlo distribuído deste exemplo de aplicação foi gerado um sinal PRBS (*PseudoRandom Binary Sequence*) com valores entre 45 e 55. Cada valor desta sequência foi usado como um patamar diferente para o sinal de referência do sistema de controlo. Foi usado um sinal de referência com 5 patamares diferentes com uma duração de 10 minutos cada. O controlo de temperatura do subsistema depósito (Apêndice A) foi deixado a cargo do termostáto que o módulo

PCT 13 dispõe. Este faz um controlo do tipo ON/OFF e foi regulado para manter uma temperatura de aproximadamente 85°C. Os parâmetros do controlador utilizados são apresentados na Tabela 4.1.

Parâmetro do controlador	**Valor**
Período de amostragem (h)	10s
Pólo em malha fechada (α_m)	0.05rad/s
Pólo do observador (α_O)	0.1rad/s
Factor de esquecimento (λ)	0.99

Tabela 4.1: Parâmetros do controlador

4.1.5 Resultados

Na Figura 4.3 são apresentados os sinais relativos aos testes efectuados sob as condições anteriormente referidas. Nesta estão incluídos os sinais de referência e de saída, o sinal de controlo e o sinal de erro.

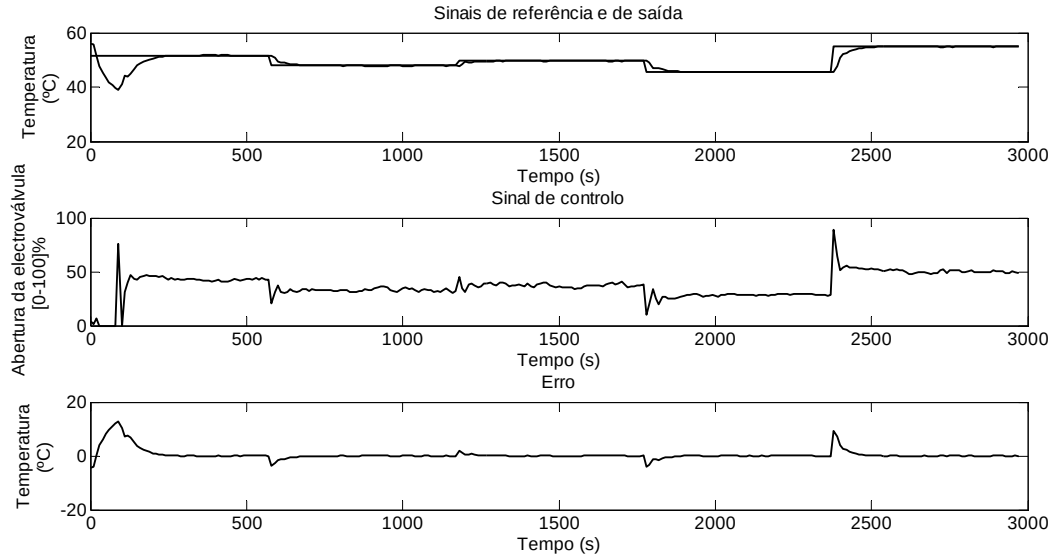


Figura 4.3: Sinais relativos ao teste do exemplo de aplicação 1

Relativamente aos resultados apresentados na Figura 4.3, verifica-se que o sinal de saída consegue atingir os valores de patamares impostos pelo sinal de referência e acompanhá-los com um erro bastante reduzido. Isto pode ser observado quer pela comparação directa dos sinais de referência e de saída quer pelo sinal de erro que se encontra muito perto de zero. O sinal de erro apresenta apenas valores mais elevados quando ocorrem transições, tal como esperado, o que está relacionado com a própria

dinâmica do sistema. Este apresenta em média um valor de 0.0417°C. De notar que este valor foi calculado a partir do instante 250s, altura em que se considerou que os parâmetros identificados já tinham estabilizado e que o sinal de saída já tinha atingido o primeiro patamar, não entrando assim em conta com as condições iniciais que iriam falsear a avaliação deste valor.

Relativamente ao sinal de controlo, este apresenta apenas pequenas oscilações e picos de maior valor quando ocorrem transições de patamar, o que era igualmente esperado. O facto de o sinal de controlo se manter praticamente estável durante cada patamar é um bom indicador da correcta parametrização do controlador. Indica que a sua dinâmica é adequada, não actuando de uma forma muito rápida ou muito lenta, o que resultaria numa oscilação do sinal de controlo.

Na Figura 4.4 é apresentada a evolução dos parâmetros identificados e a variância dos parâmetros estimados.

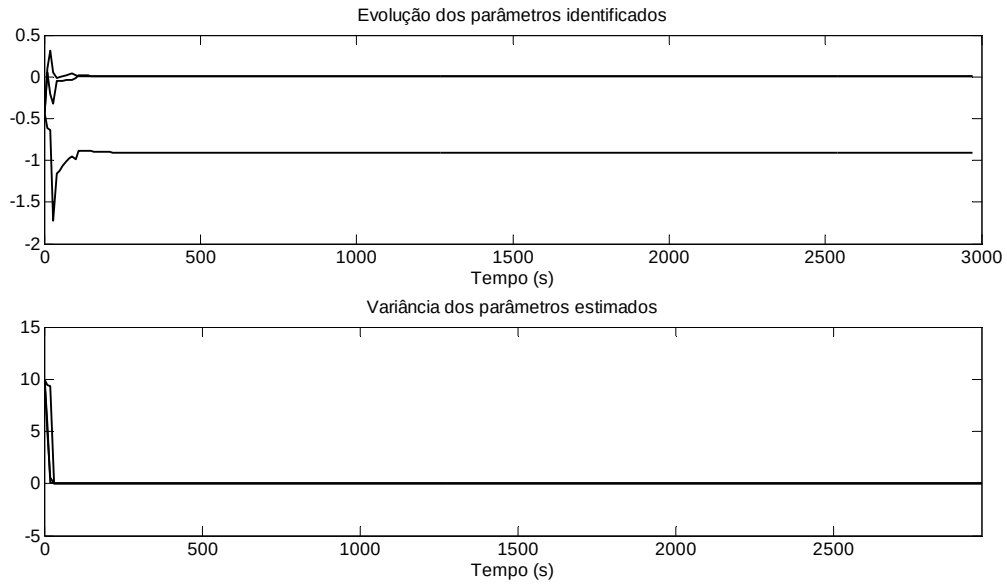


Figura 4.4: Identificação dos parâmetros relativa ao teste do exemplo de aplicação 1

Como se pode observar na Figura 4.4 os parâmetros identificados estabilizaram, o que indica que o sistema foi bem identificado. A oscilação que estes apresentam no início do teste é normal e deve-se ao facto de o estimador de tempo-real ainda não ter informação suficiente do sistema para poder identificá-lo correctamente. Da variância dos parâmetros verifica-se uma oscilação inicial que se deve à adaptação inicial dos parâmetros. De seguida verifica-se uma estabilização da mesma em torno do valor zero, o que é indicador de fiabilidade dos parâmetros estimados. Os parâmetros identificados são indicados na Tabela 4.2.

Parâmetro	Valor
$\hat{\theta}_1$	-0.9100
$\hat{\theta}_2$	0.0066
$\hat{\theta}_3$	0.0088

Tabela 4.2: Parâmetros identificados

Adicionalmente, verificou-se que o tempo de execução do algoritmo de controlo usando o MATLAB é em média de 0.339s, valor este bastante abaixo do tempo de amostragem, que se encontra com o valor de 10s.

4.2 Controlo de processo térmico baseado em Scilab e Java

Este exemplo de aplicação visa, igualmente, o controlo da temperatura do fluxo de água de saída do módulo PCT 13 da Armfield, apresentado no Apêndice A. No entanto, ao contrário do exemplo de aplicação anterior, é utilizado o Scilab em vez do MATLAB seguindo, assim, a filosofia de comunicação apresentada na secção 3.6. O controlador é, portanto, totalmente implementado em Scilab e a comunicação com o exterior é feita pela acesso à *gateway* intermédia apresentada na secção 3.6.1. A interface com sensores e actuadores é igualmente feita remotamente no acesso a uma Mini-ITX D945GCLF da Intel que corre a *gateway* remota descrita na secção 3.5.1.

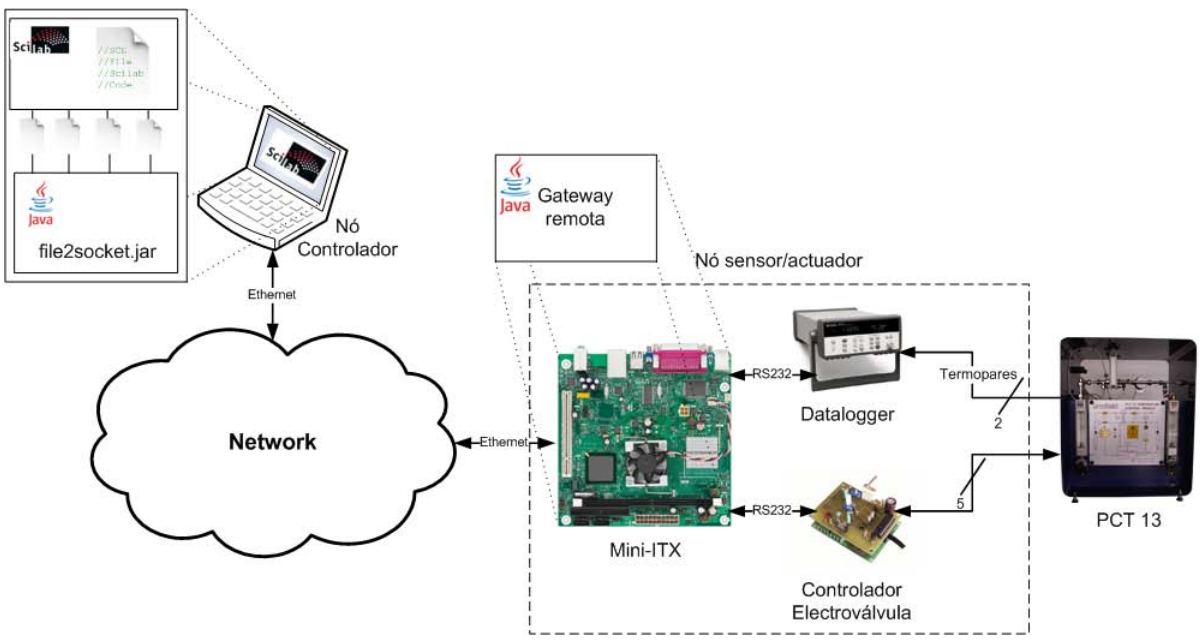


Figura 4.5: Exemplo de aplicação 2

4.2.1 Estrutura do sistema

O sistema de controlo distribuído deste exemplo de aplicação segue a estrutura apresentada na Figura 4.5. Tal como pode ser observado, este exemplo de aplicação utiliza uma rede de comunicação do tipo *Ethernet*. Mais uma vez o nó controlador é composto por um PC com grande capacidade de cálculo. É neste PC que reside o Scilab e a *gateway* intermédia. O nó sensor/actuador é composto pelo PC Mini-ITX, um *datalogger* HP34970A e uma placa de controlo de electroválvula. A interacção da Mini-ITX com o datalogger e com controlador de electroválvula é feita via RS232. O *datalogger* permite a leitura dos termopares presentes no módulo PCT 13 de uma forma simplificada, usando para o efeito a norma SCPI. O controlador da electroválvula permite um controlo simplificado da posição de abertura da electroválvula, bastando o envio de um *setpoint* via RS232.

4.2.2 Algoritmo de controlo

O algoritmo de controlo utilizado é o algoritmo de controlo adaptativo por posicionamento de pólos com tempo morto fraccionário utilizado no exemplo de aplicação anterior.

4.2.3 Condições de teste

As condições de teste são as mesmas que no exemplo de aplicação anterior descritas na secção 4.1.4. O sinal de referência utilizado é também o mesmo que o utilizado no exemplo de aplicação anterior.

4.2.4 Resultados

Na Figura 4.6 são apresentados os sinais relativos aos testes efectuados sob as condições anteriormente referidas. Nesta estão incluídos os sinais de referência e de saída, o sinal de controlo e o sinal de erro.

Relativamente aos resultados apresentados na Figura 4.6, verifica-se que o sinal de saída consegue atingir os valores de patamares impostos pelo sinal de referência e acompanhá-los com um erro bastante reduzido. Isto pode ser observado quer pela comparação directa dos sinais de referência e de saída quer pelo sinal de erro que se encontra muito perto de zero. O sinal de erro apresenta apenas valores mais elevados quando ocorrem transições, tal como esperado, o que, tal como anteriormente mencionado, está relacionado com a própria dinâmica do sistema. Este apresenta em média

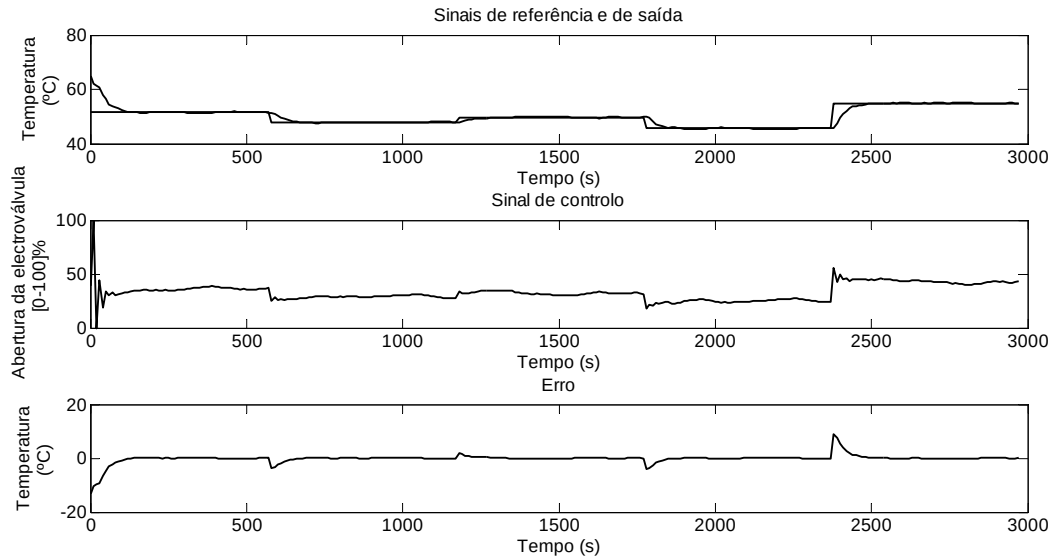


Figura 4.6: Sinais relativos ao teste do exemplo de aplicação 2

um valor de 0.0466°C. De notar que este valor foi calculado a partir do instante 250s, altura em que se considerou que os parâmetros identificados já tinham estabilizado e que o sinal de saída já tinha atingido o primeiro patamar, não entrando assim em conta com as condições iniciais que iriam falsear a avaliação deste valor.

Relativamente ao sinal de controlo, este apresenta apenas pequenas oscilações e picos de maior valor quando ocorrem transições de patamar, o que era igualmente esperado. O facto de o sinal de controlo se manter praticamente estável durante cada patamar é um bom indicador da correcta parametrização do controlador. Indica que a sua dinâmica é adequada, não actuando de uma forma muito rápida ou muito lenta, o que resultaria numa oscilação do sinal de controlo.

Na Figura 4.7 é apresentada a evolução dos parâmetros identificados e a variância dos parâmetros estimados.

Como se pode observar na Figura 4.7 os parâmetros identificados estabilizaram, o que indica que o sistema foi bem identificado. A oscilação que estes apresentam no início do teste é normal e deve-se ao facto de o estimador de tempo-real ainda não ter informação suficiente do sistema para poder identificá-lo correctamente. Da variância dos parâmetros verifica-se uma oscilação inicial que se deve à adaptação inicial dos parâmetros. De seguida verifica-se uma estabilização da mesma em torno do valor zero, o que é indicador de fiabilidade dos parâmetros estimados. Os parâmetros identificados são semelhantes aos obtidos no exemplo de aplicação anterior apresentados na tabela 4.2.

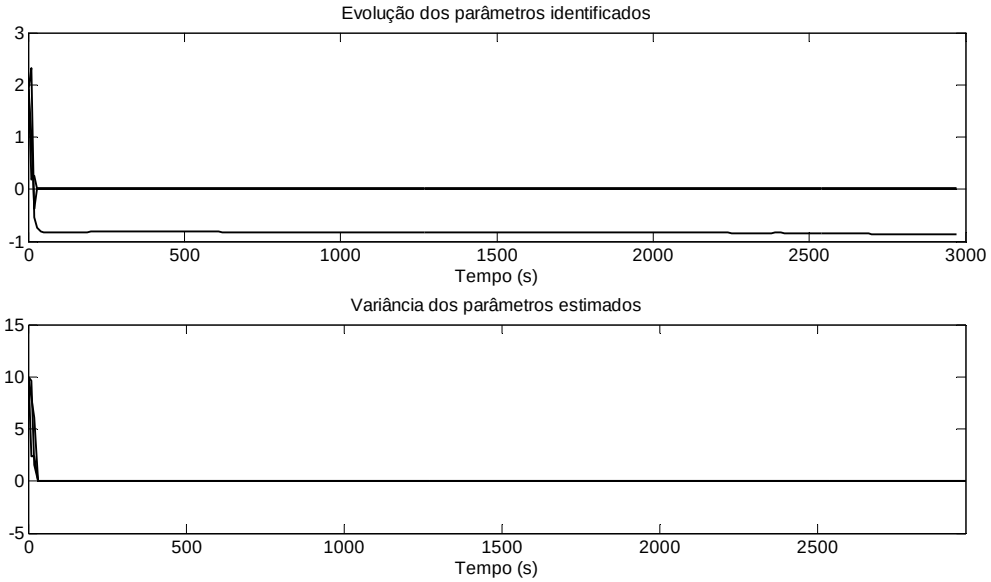


Figura 4.7: Identificação dos parâmetros relativa ao teste do exemplo de aplicação 2

Adicionalmente, verificou-se que o tempo de execução do algoritmo de controlo usando o Scilab é em média de 0.312s, valor este bastante abaixo do tempo de amostragem, que se encontra com o valor de 10s.

4.3 Sistema de aquisição baseado em barramentos 1-Wire, Java e MATLAB

Neste exemplo de aplicação é apresentado um sistema de aquisição usando sensores do tipo 1-Wire. Para o efeito é utilizado, conjuntamente, linguagem Java, MATLAB e barramentos do tipo 1-Wire. O sistema de aquisição é distribuído sobre uma rede *Ethernet* e utiliza *sockets* TCP implementados em linguagem Java. É utilizada a API desenvolvida no âmbito desta dissertação, denominada de javaOnMatlab.jar, que é incluída no caminho de classes do MATLAB e que inclui um objecto, denominado de owAdapter, que estende as funcionalidades disponibilizadas pela 1-Wire API para Java da Maxim.

4.3.1 Estrutura do sistema

Este exemplo de aplicação segue a estrutura apresentada na Figura 4.8. Neste caso temos uma Mini-ITX da Intel ligada a uma rede *Ethernet* que corre um servidor do tipo 1-Wire NetAdapter. Este servidor virtualiza o barramento 1-Wire na rede por

forma a que os métodos disponibilizados pela 1-Wire API para Java possam ser executados normalmente como se o barramento se encontrasse ligado localmente ao PC. À Mini-ITX encontra-se ligado um conversor de 1-Wire para RS232 ou USB, tal como o apresentado no Apêndice D, que permite o acesso ao barramento.

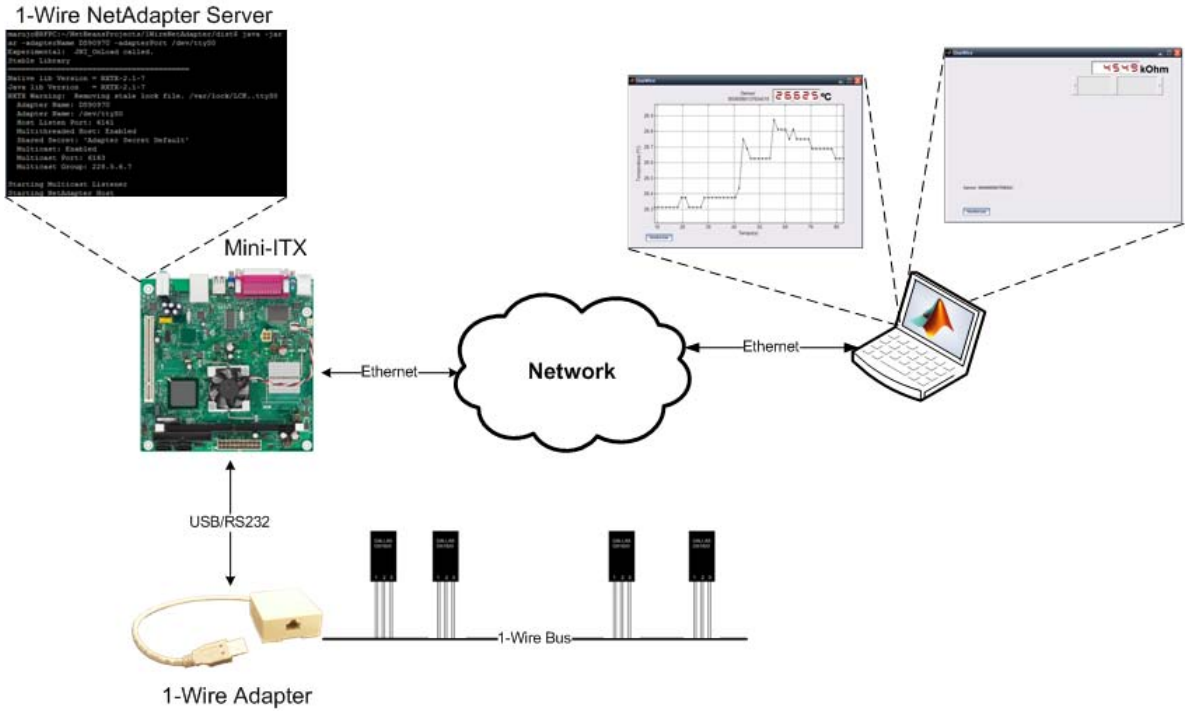


Figura 4.8: Exemplo de aplicação 3

Com a inclusão da API javaOnMatlab.jar no MATLAB torna-se fácil a monitorização dos sensores presentes no barramento remoto a partir do mesmo e, potencialmente, até de vários barramentos distribuídos numa rede da mesma forma. Para teste e demonstração do conceito foi criada uma aplicação com ambiente gráfico em MATLAB que permite uma fácil monitorização dos sensores. Utilizando esta aplicação devemos primeiro indicar o ip e porto do PC remoto, tal como demonstrado na Figura 4.9.

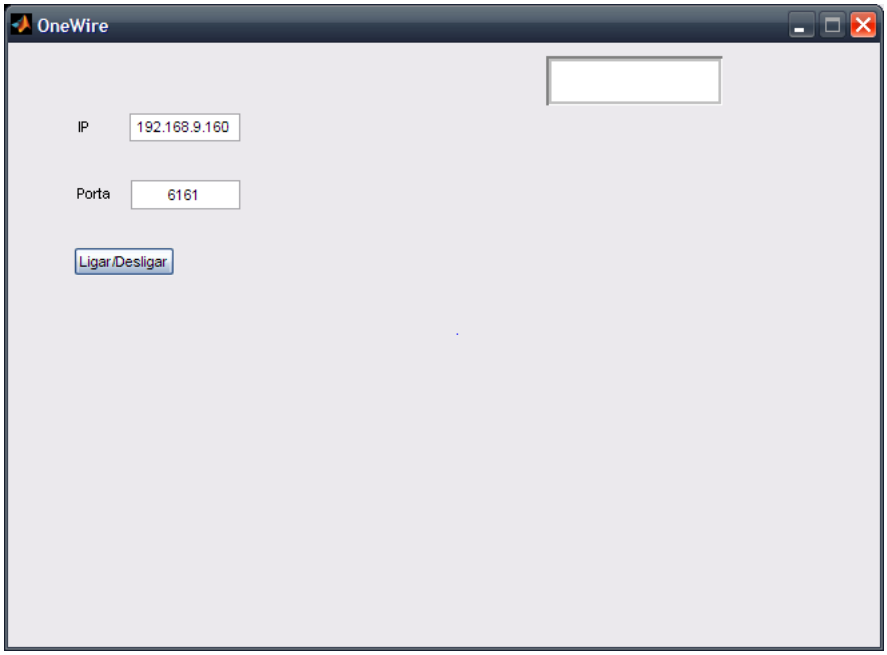


Figura 4.9: Indicação de IP e porto

Após a ligação ser efectuada é disponibilizada uma listagem dos sensores presentes no barramento remoto apresentando as suas chaves de fábrica, tal como demonstrado na Figura 4.10.

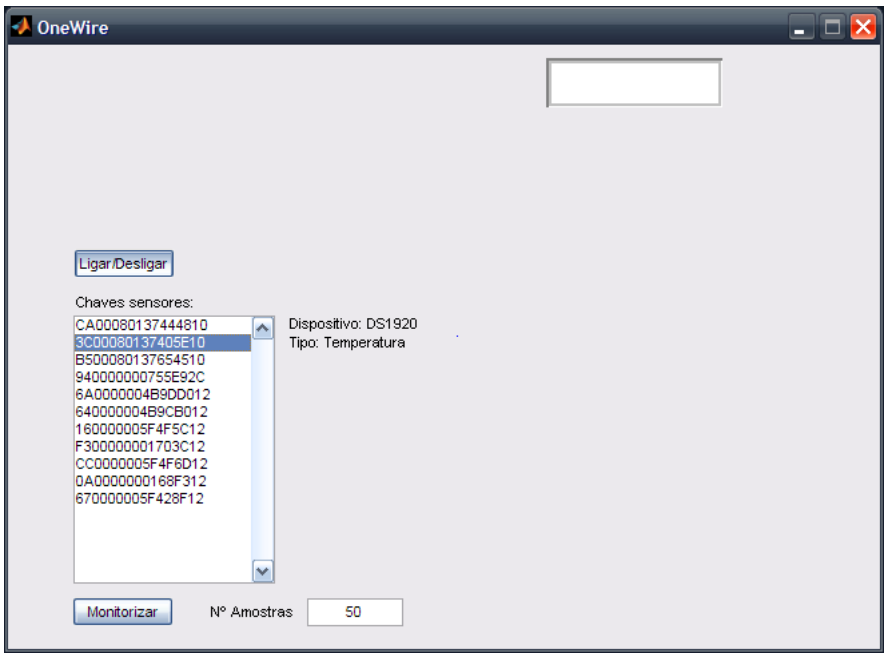


Figura 4.10: Listagem de sensores

Seleccionando, por exemplo, um sensor de temperatura é possível monitorizar a sua actividade em tempo real, tal como demonstrado na Figura 4.11.

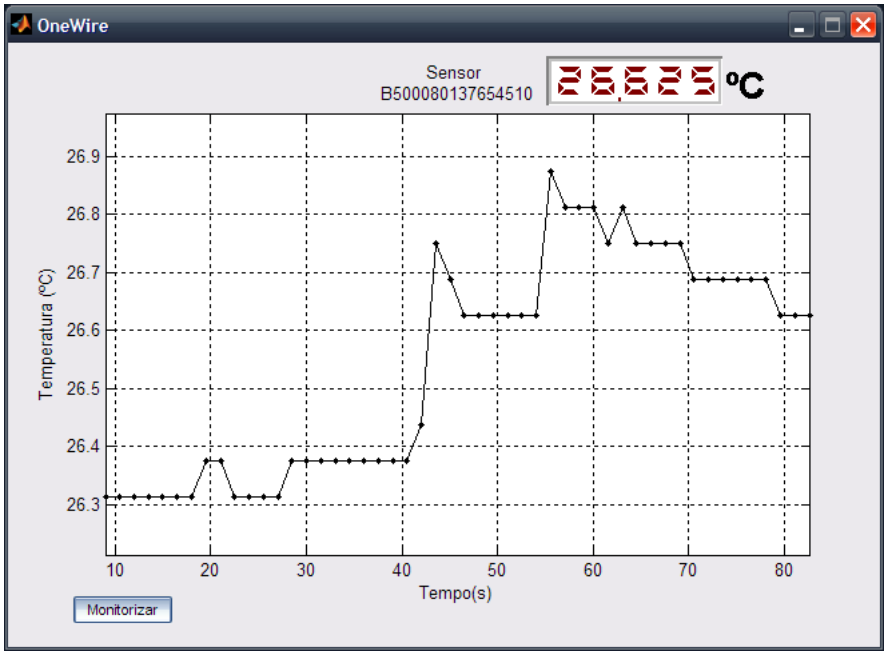


Figura 4.11: Monitorização de sensor de temperatura

Seleccionando um *switch* é possível monitorizar o seu estado e alterá-lo, e seleccionando um potenciómetro digital é possível monitorizar e alterar o valor de resistência que apresenta, tal como demonstrado na Figura 4.12 e Figura 4.13, respectivamente.

Figura 4.12: Monitorização de um switch

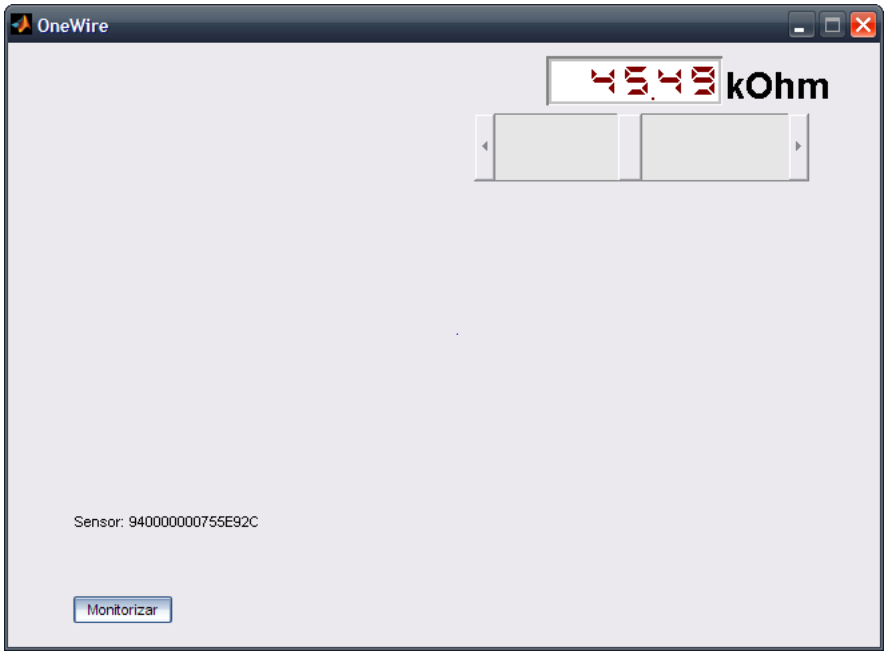


Figura 4.13: Monitorização de um potenciómetro digital

Estes são apenas alguns exemplos que esta aplicação suporta, podendo muito facilmente serem adicionados mais alguns visando cobrir uma maior variedade de sensores. Adicionalmente, a estrutura presente neste exemplo de aplicação poderia ser alterada para efectuar, inclusivamente, controlo tendo sempre em conta as limitações impostas pela velocidade de acessos a barramentos deste tipo e as limitações temporais da linguagem Java. Este tipo de estrutura poderia ser facilmente utilizado na monitorização e controlo de, por exemplo, estufas ou estações metereológicas.

Capítulo 5

Conclusões e trabalho futuro

5.1 Conclusões

Este trabalho foca essencialmente a utilização da linguagem Java na criação de sistemas de controlo distribuídos sobre redes do tipo *Ethernet*. Para o efeito é feita uma análise detalhada de várias filosofias de comunicação. Várias soluções são expostas e analisadas.

Para facilidade de implementação de algoritmos de controlo com um grau de complexidade mais elevado é introduzida a possibilidade da utilização conjunta da linguagem Java com programas de computação numérica. A utilização de programas deste tipo facilitou, igualmente, o manuseamento de informação e a sua análise.

No âmbito desta dissertação foram desenvolvidas com sucesso malhas de controlo distribuídas que fazem uso da linguagem Java e dos programas MATLAB e Scilab. Estas malhas visaram o controlo do processo térmico presente no módulo PCT 13 e fizeram uso de um algoritmo de controlo adaptativo.

Adicionalmente foram criadas infraestruturas para um sistema de aquisição remoto baseado na utilização conjunta de barramentos do tipo 1-Wire, linguagem Java e MATLAB.

Todas estas soluções foram implementadas e testadas correndo em plataformas *Windows* e *Linux*.

5.2 Trabalho futuro

Neste ponto são feitas algumas sugestões quanto a trabalho futuro que se julga serem elementos complementares importantes ao trabalho desenvolvido no âmbito desta dissertação.

5.2.1 Criação de servidor *multi-threaded*

Vários são os programas criados no âmbito desta dissertação que utilizam servidores TCP. É caso disto a *gateway* remota apresentada na secção 3.5.1. Esta utiliza um servidor com um único fio de execução, o que limita o acesso exclusivo de um cliente de cada vez. A modificação deste servidor por forma a permitir o acesso a múltiplos clientes em simultâneo, juntamente com um mecanismo de gestão de recursos, permitiria um melhor aproveitamento dos recursos locais virtualizados na rede por esta *gateway*.

5.2.2 Utilização de RMI (*Remote Method Invocation*)

A utilização de RMIs parece ser, igualmente, algo a explorar. Isto permitiria, por exemplo, que na utilização do MATLAB se pudessem fazer chamadas a métodos residentes num servidor remoto. Para além de evitar uma constante actualização das classes e a sua inclusão no MATLAB, permitiria também criar um servidor de classes central que seria o único a actualizar, ao qual todos os nós poderiam aceder e cujas classes seriam para os demais nós automaticamente actualizadas.

5.2.3 Adição de novas funcionalidades à *gateway* remota

Poderiam ser adicionadas novas funcionalidades à *gateway* remota, tais como suporte a novos tipos de interfaces e construção de uma gama de comandos mais alargada.

Apêndice A

Módulo de controlo de temperatura PCT 13

Luís Farinha, Nuno Marujo, Rui Sancho
Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro
Maio 2009

(a) Visão frontal (b) Visão traseira

Figura A.1: Módulo PCT 13

O módulo de controlo de temperatura PCT 13 é um equipamento destinado ao ensino e investigação, desenvolvido pela Armfield, que faz parte de uma alargada gama de produtos que a mesma possui e que visam demonstrar processos de medição e controlo.

Este módulo tem por objectivo o controlo de um processo térmico, que consiste no aquecimento de um caudal de água de entrada até a uma temperatura desejada. Para este efeito, o módulo possui um pequeno depósito de água, situado à retaguarda, que contém no seu interior uma resistência. Quando se aplica uma corrente à resistência, esta dissipa energia aquecendo, assim, a água contida no depósito. No topo do depósito está situada uma bomba que faz circular a água contida no mesmo, através de um circuito fechado. O caudal que circula neste pode ser regulado por uma válvula manual ou por uma electroválvula.

Na retaguarda do módulo está situado um tubo por onde pode ser injectado um caudal de água fria. Este caudal pode, igualmente, ser regulado por uma válvula manual. Quando injectada, a água flui por um circuito que passa paralelamente ao circuito fechado de água quente através de um permutador, saindo finalmente por um tubo na lateral do módulo.

O permutador é um elemento chave do sistema, propiciando a transferência de energia entre o caudal de água quente e o caudal de água fria. É deste modo que o caudal de água fria é aquecido à temperatura de saída desejada.

O módulo inclui um controlo ON/OFF que permite controlar a temperatura da água no depósito, sendo necessário, para o efeito, regular um botão que se encontra na caixa à retaguarda.

Painel de interface ao módulo PCT 13

O módulo de controlo de temperatura PCT 13 possui um painel de interface, cuja imagem está apresentada na Figura A.2.

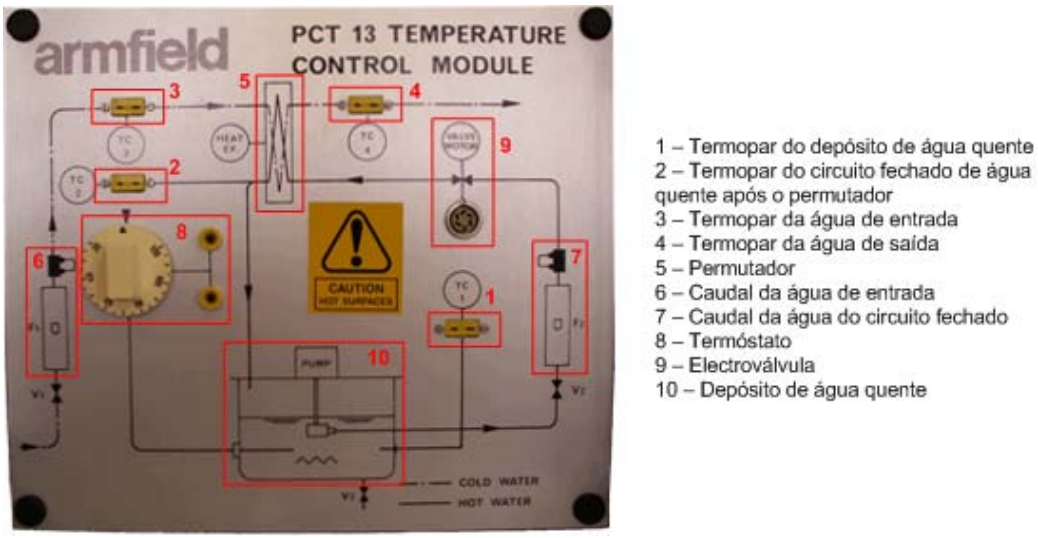


Figura A.2: Painel de interface

- **Termopares** - Para a aquisição da temperatura nos diferentes pontos do PCT 13, são usados 4 termopares do tipo K (1, 2, 3 e 4). O termopar 1 permite efectuar a leitura da temperatura presente no depósito de água quente; o termopar 2 permite efectuar a leitura da temperatura do circuito fechado de água quente, após a passagem pelo permutador; os termopares 3 e 4 permitem efectuar a leitura da temperatura da água de entrada e de saída, respectivamente.

- **Permutador** - O módulo possui um permutador (5) que, como já foi referido anteriormente, faz a transferência de energia da água quente para a água fria, resultando assim na saída a temperatura da água desejada.

- **Depósito de água quente -** A temperatura da água no circuito fechado é regulada com base na temperatura presente no depósito (10). Este possui internamente uma resistência pela qual, a água é aquecida. A temperatura da água pode ser controlada por um termostáto (8), ou então, por controlo de potência baseado na técnica de contagem de impulsos de rede.

- **Termostáto** - O termostáto (8), como foi referido, possibilita controlar de forma ON/OFF a temperatura no depósito de água quente.

- **Caudal de água** - Existem dois caudais presentes no PCT 13 que influenciam directamente a temperatura de saída: um para a entrada de água fria (6) e outro para a água quente (7). O primeiro caudal indica o volume de água de entrada no PCT 13 e é controlado de forma manual; o segundo indica o volume de água que circula no circuito fechado, onde pode ser controlado de duas formas: de forma manual ou através de uma electroválvula (9).

- **Electroválvula** - A electroválvula (9) permite determinar o volume de água que circula no circuito fechado, possibilitando, com base na sua posição, controlar a temperatura da água de saída.

Controlo do módulo PCT 13

Por forma a controlar o módulo PCT 13 foi necessária a adição de alguns módulos ao sistema, seguindo a estrutura presente na Figura A.3.

A temperatura de saída é influenciada por diversas variáveis, tais como a temperatura no depósito, o caudal de entrada e a posição da electroválvula. O sistema permite a medição da temperatura em diversos pontos através dos termopares que disponibiliza. Deste modo, várias filosofias de controlo podem ser adoptadas.

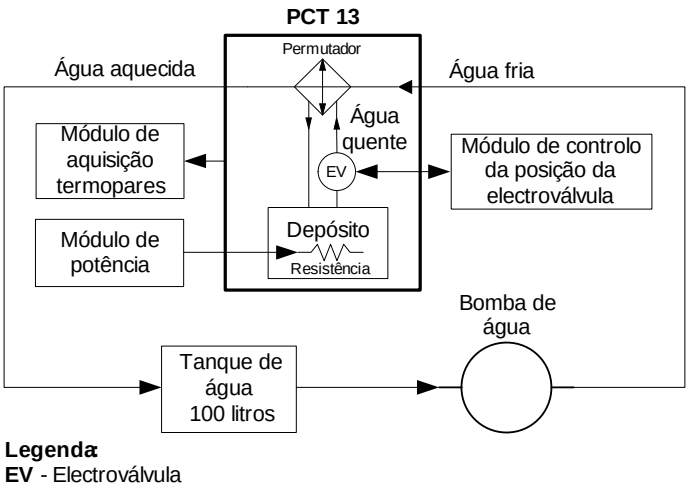


Figura A.3: Sistema de controlo módulo PCT 13

Os módulos adicionados ao sistema visam o controlo da temperatura da água no depósito, da posição da electroválvula e da aquisição de sinal dos termopares para monitorização do sistema e foram utilizados para efeitos de caracterização da dinâmica do mesmo.

Módulo de potência

O módulo de potência permite o controlo da potência fornecida à resistência de aquecimento, presente no depósito, numa determinada janela temporal. O módulo utilizado é apresentado na Figura A.4.

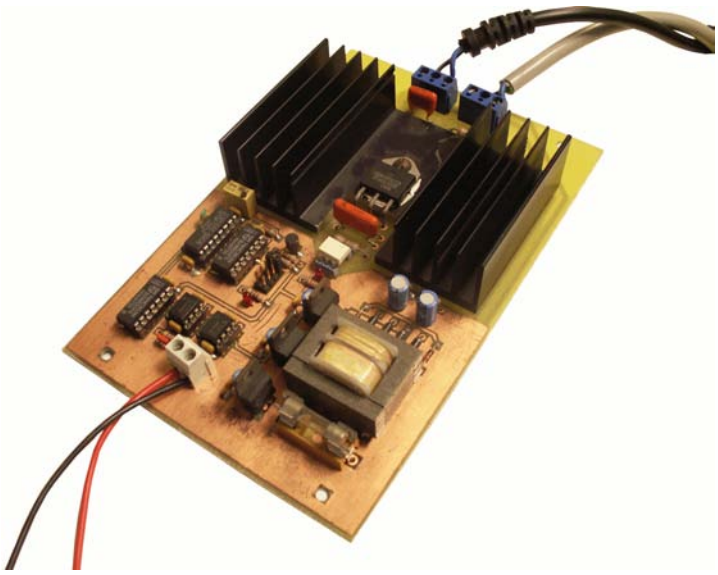


Figura A.4: Módulo de potência

Esta placa aplica um controlo por ciclos de rede, ou seja, numa determinada janela temporal define em quantos ciclos de rede é aplicada potência à resistência. Desta forma, consegue definir uma potência média durante esse intervalo.

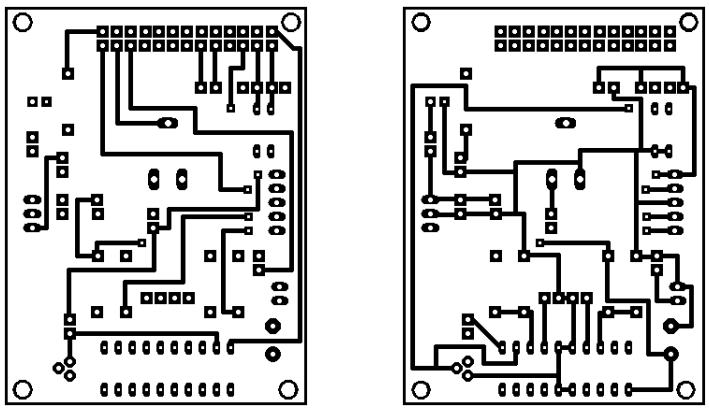
Por forma a calcular a potência máxima que pode ser fornecida à resistência (caso que acontece quando conduz todos os ciclos) foi necessário medir o valor da resistência. Esta apresenta um valor de $57,2\Omega$, o que implica uma potência dissipada de aproximadamente 1000W.

Módulo de controlo da electroválvula

O módulo de controlo da electroválvula permite controlar o posicionamento da mesma, e consequentemente a temperatura da água de saída. Este módulo é apresentado na Figura A.5.

Figura A.5: Controlador da electroválvula

Este módulo possui quatro interfaces distintas. Os acessos podem ser feitos via RS232, barramento CAN, potenciómetro manual ou potenciómetro externo. O controlo da electroválvula é feito, essencialmente, através de um microcontrolador e de uma ponte-H. Este módulo é composto por duas placas, uma placa DETPIC desenvolvida pelo Departamento Electrónica, Telecomunicações e Informática da Universidade de Aveiro e uma placa de expansão, desenvolvida no âmbito desta dissertação, cujo projecto resultou no esquema de circuito impresso apresentado na Figura A.6.



(a) Frente (b) Traseira

Figura A.6: Circuito impresso placa electroválvula

O esquema eléctrico correspondente a esta placa é apresentado na Figura A.7.

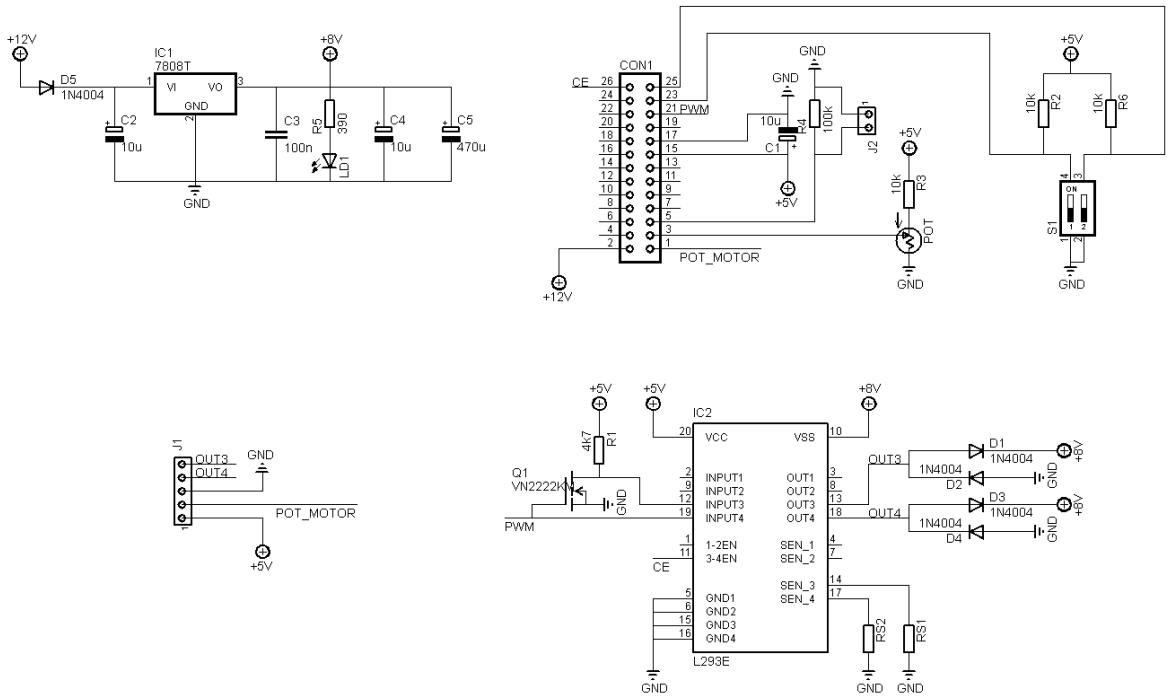


Figura A.7: Esquema eléctrico placa Electroválvula

Módulo de aquisição termopares

Para o módulo de aquisição dos termopares é necessário um equipamento de instrumentação que acondicione o sinal. Para efeitos de caracterização do sistema optou-se pela utilização do equipamento de instrumentação HP34970A da Hewlett-Packard, que possibilita a interface com termopares do tipo K, fazendo a amplificação do sinal e a

compensação de junção fria. Esta opção possui ainda a vantagem deste aparelho suportar a norma SCPI, o que permite que se possa monitorizar o estado dos termopares no PC via RS232. Este equipamento é apresentado na Figura A.8.

Figura A.8: Equipamento de instrumentação HP34970A

Bomba de água

Para a injecção de água no sistema foi necessário a utilização de uma bomba, que bombeasse a água armazenada num depósito para o módulo PCT 13. A bomba utilizada é apresentada na Figura A.9.

Figura A.9: Bomba de água

Depósito de água

Por forma a que fosse possível a execução dos testes necessários sem um gasto despropositado de água, utilizou-se um depósito de 100 litros, do qual a água era bombeada e para o qual era reposta. O depósito é apresentado na Figura A.10.

Figura A.10: Depósito de água

Caracterização do sistema

O sistema apresentado pode ser separado em dois subsistemas e, por conseguinte, podem ser efectuadas duas caracterizações distintas. Numa primeira abordagem será caracterizado o subsistema depósito onde será apresentada a sua resposta ao degrau. Numa fase posterior será apresentada a resposta ao degrau da electroválvula.

Subsistema depósito

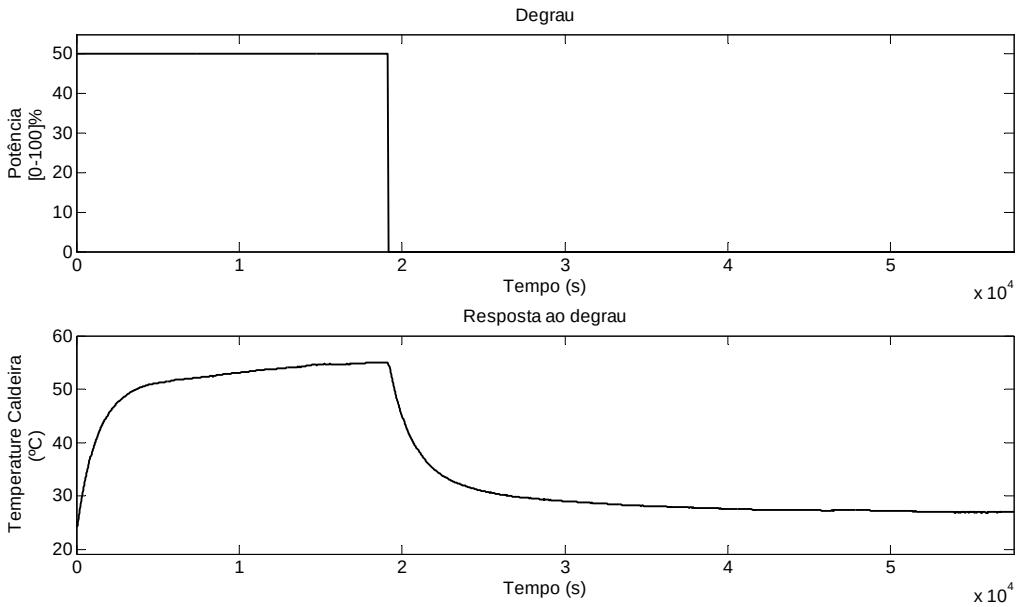


Figura A.11: Resposta ao degrau

Para a caracterização do subsistema depósito utilizou-se o módulo de potência anteriormente apresentado. Partindo da temperatura ambiente, aplicou-se um degrau com 50% de potência à resistência interna do depósito e monitorizou-se a temperatura do

depósito. Quando esta estabilizou, deixou-se de aplicar potência à resistência, deixando assim que a água arrefecesse. Esta experiência permitiu obter a constante de tempo de aquecimento e arrefecimento do subsistema depósito. Os resultados obtidos são apresentados na Figura A.11.

Através dos resultados obtidos concluiu-se que as constantes de tempo de aquecimento e arrefecimento são aproximadamente de 1650s e 3400s, respectivamente.

Subsistema electroválvula

O subsistema electroválvula consiste no controlo do caudal de água quente através do posicionamento da electroválvula, por forma a que a água aquecida atinja uma determinada temperatura.

Para caracterizar o subsistema electroválvula regulou-se a temperatura da água quente para um valor de 75ºC. O controlo desta temperatura fica a cargo do termostáto presente na caixa à retaguarda do módulo PCT 13. Após esta temperatura se encontrar relativamente estável, obtém-se a resposta ao degrau aplicado na electroválvula para caudais de entrada de 50, 100, 200 e 280 cm^3/min. Através desta experiência é possível determinar quais as constantes de tempo associadas ao subsistema electroválvula, assim como o valor do tempo morto, caso este o apresente. Os gráficos seguintes apresentam a resposta ao degrau para os vários caudais.

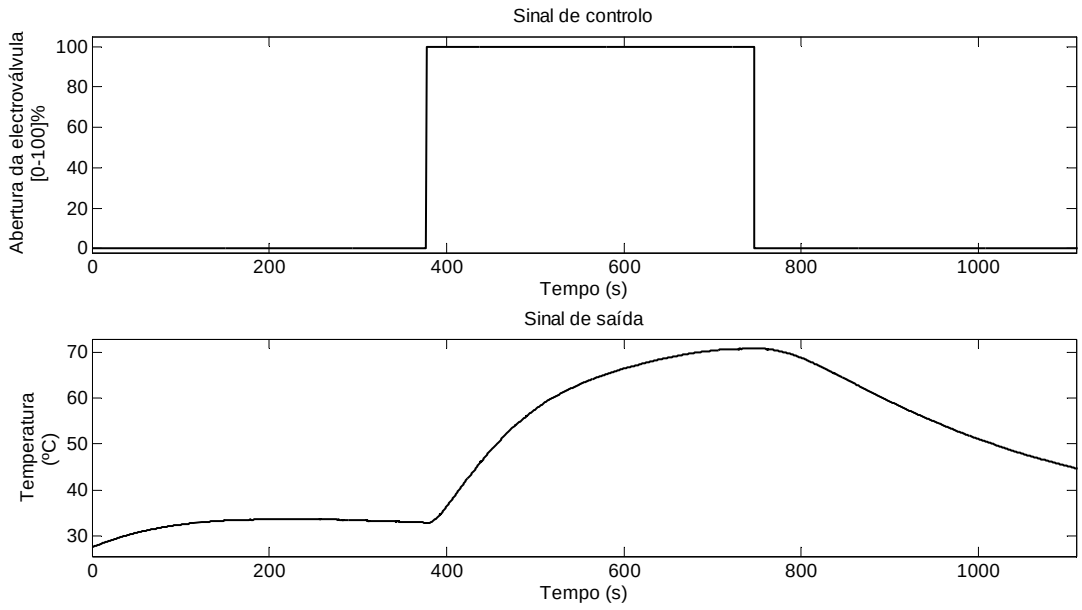


Figura A.12: Resposta ao degrau do subsistema electroválvula para um caudal de entrada de 50 cm^3/min

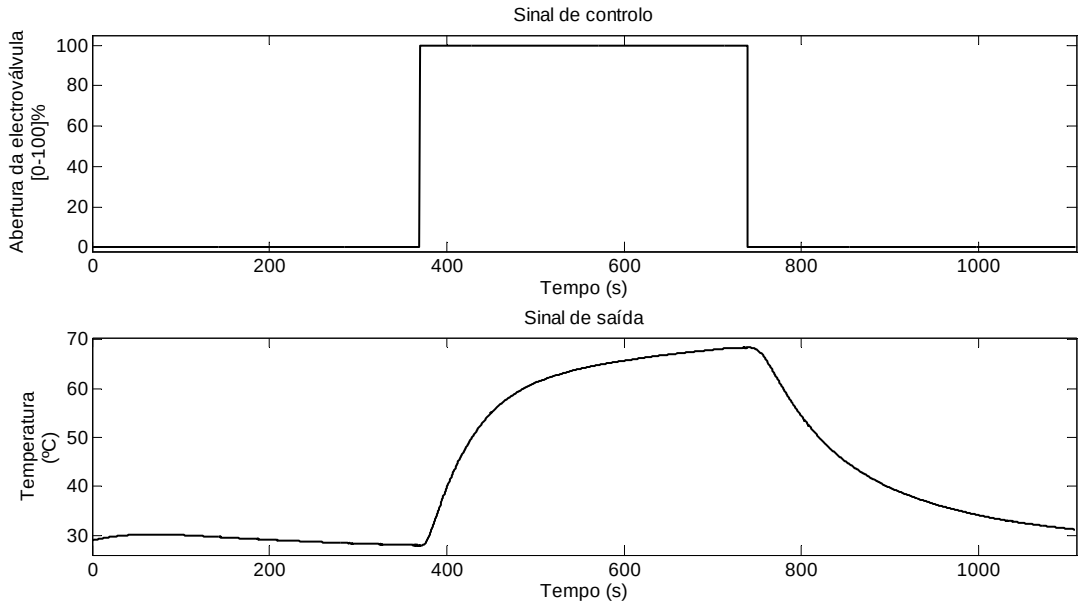


Figura A.13: Resposta ao degrau do subsistema electroválvula para um caudal de entrada de 100 cm^3/min

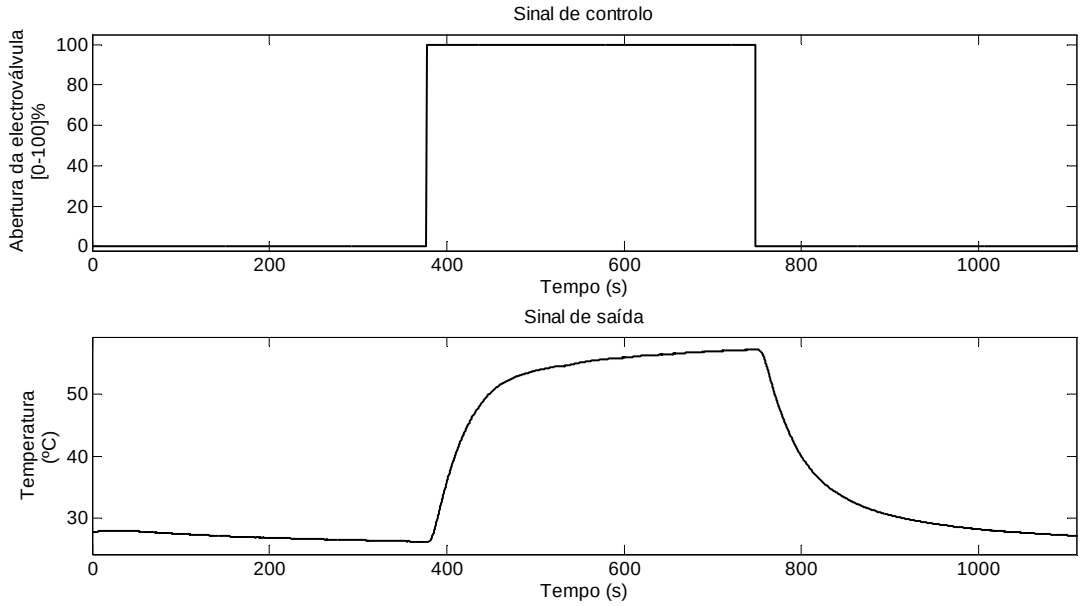


Figura A.14: Resposta ao degrau do subsistema electroválvula para um caudal de entrada de 200 cm^3/min

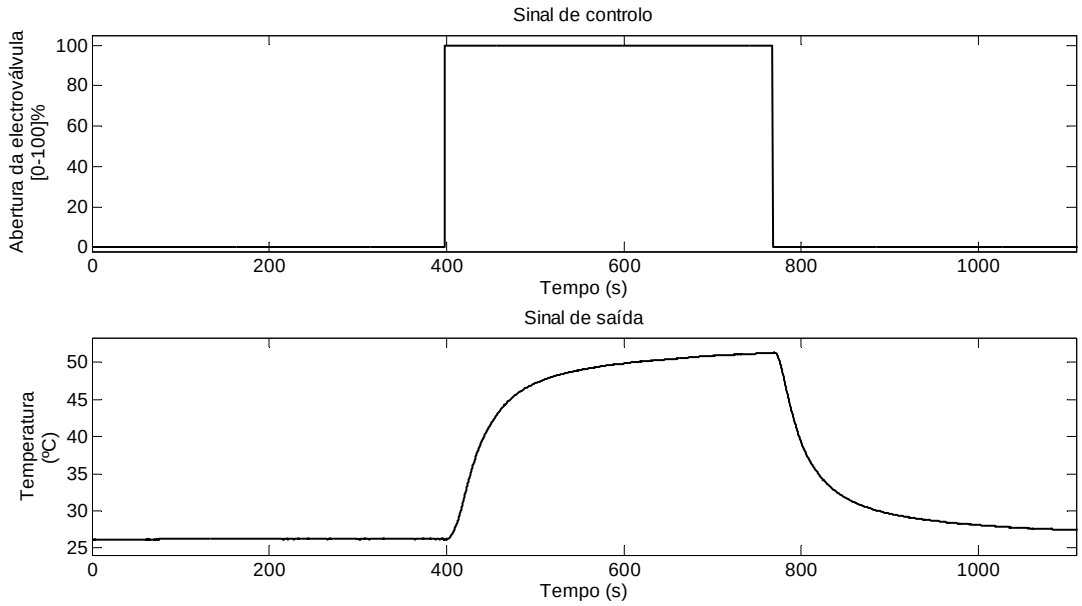


Figura A.15: Resposta ao degrau do subsistema electroválvula para um caudal de entrada de 280 cm^3/min

Análise dos resultados

Após uma análise dos resultados apresentados anteriormente, conclui-se que o tempo morto apresentado pelo subsistema electroválvula, ou seja o tempo que este demora a reagir a uma excitação externa, é de aproximadamente 4s. No que diz respeito aos tempos de subida, que correspondem ao tempo que o sinal de saída demora a atingir 63% do seu valor máximo quando excitado por um degrau, estes são apresentados na tabela A.1 juntamente com os tempos de descida.

Caudal (cm^3/min)	**Tempo de subida (s)**	**Tempo de descida (s)**
50	120.8	202.9
100	68.8	111.4
200	48.5	57.3
280	50	43.1

Tabela A.1: Constantes de tempo subsistema electroválvula

Identificação do subsistema electroválvula

Para a identificação do subsistema electroválvula gerou-se um sinal PRBS *(Pseudo-Random Binary Sequence)* que foi utilizado como sinal de controlo de abertura da electroválvula. O período de amostragem utilizado foi de 10s. Monitorizou-se a temperatura da água de saída do sistema e no fim do teste usaram-se estes valores para a

obtenção de um modelo matemático do sistema. Para este efeito utilizou-se o método dos mínimos quadrados para um sistema de 1ª ordem.

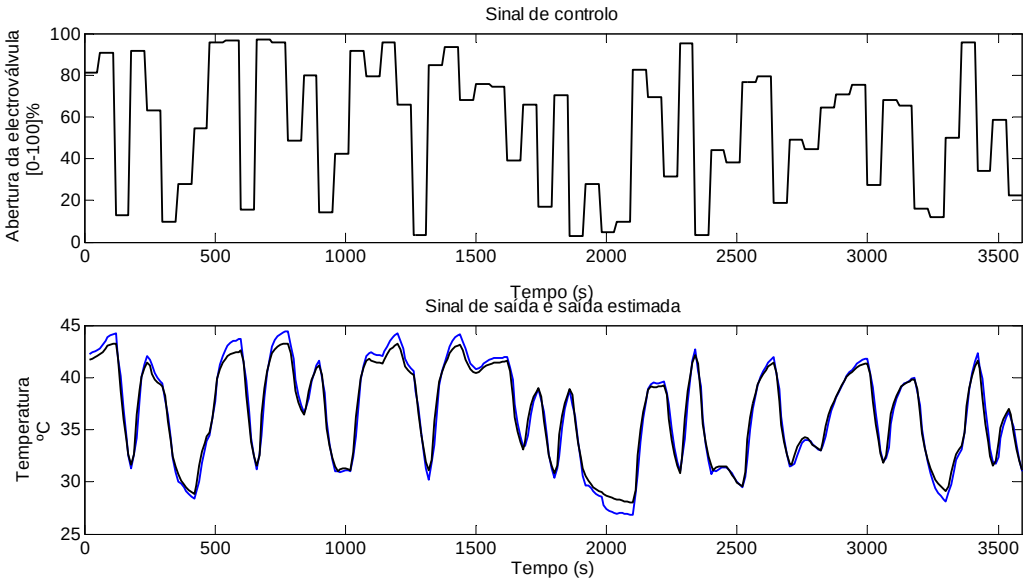


Figura A.16: Identificação subsistema electroválvula

Na Figura A.16 pode ser observado o sinal de controlo utilizado, bem como a saída do sistema e a saída estimada obtida usando o modelo matemático estimado. Verifica-se que estes dois sinais se assemelham muito, o que indica que o modelo do sistema é bastante preciso. O modelo matemático obtido é apresentado na expressão A.1.

$$G(z) = \frac{\hat{\theta}_2 z^{-1} + \hat{\theta}_3 z^{-2}}{1 + \hat{\theta}_1 z^{-1}} = \frac{0.0057 z^{-1} + 0.0017 z^{-2}}{1 - 0.9430 z^{-1}} \tag{A.1}$$

Apêndice B

Inclusão de classes no MATLAB

Uma ferramenta muito utilizada no âmbito desta dissertação é a inclusão de objectos Java no ambiente do MATLAB por forma a estender funcionalidades do mesmo. Para isto torna-se necessário que após a compilação do código e a respectiva passagem para um pacote do tipo *Java Archive* este seja incluído no caminho de classes do MATLAB.

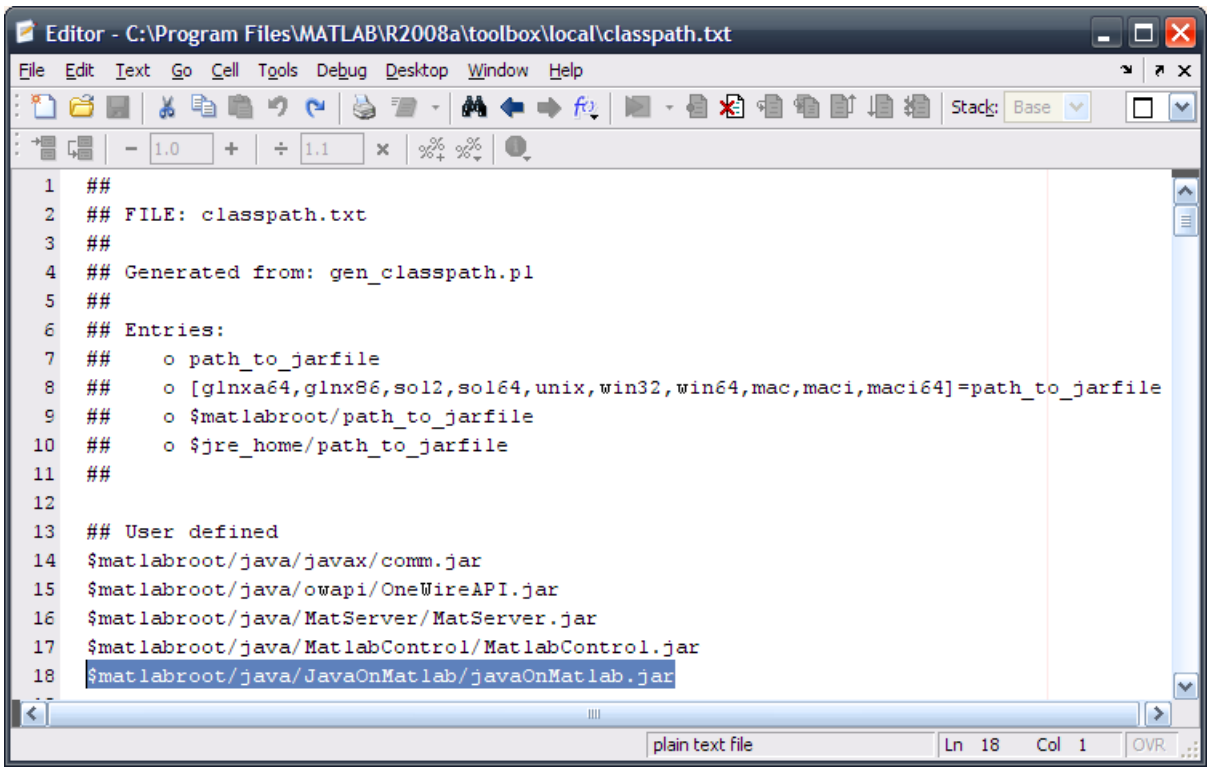


Figura B.1: Inclusão no caminho de classes

Isto deve-se ao facto de ser no arranque que o MATLAB vai ao seu caminho de classes, verifica a listagem e carrega as que lá constarem. Uma classe não carregada no processo de inicialização do MATLAB não é conhecida pelo mesmo e, consequente-

mente, não pode ser utilizada.

Para incluir uma classe no caminho de classes do MATLAB é necessário que primeiro se defina uma pasta onde esta deva constar. Por exemplo, nos testes efectuados, a API usada foi inserida na raíz de instalação do MATLAB sob o directório *$matlabroot/java/JavaOnMatlab/*. Após isto o ficheiro *classpath.txt* deve ser editado evocando a partir da consola do MATLAB o comando *edit 'classpath.txt'*. Aberto o ficheiro adiciona-se uma linha com o caminho para o *Java Archive*. No caso utilizado, adicionou-se *$matlabroot/java/JavaOnMatlab/javaOnMatlab.jar*, tal como apresentado na Figura B.1. Seguidamente basta salvar o ficheiro, reiniciar o MATLAB e a partir daí este já conhecerá as classes adicionadas.

Apêndice C

API desenvolvida: javaOnMatlab

Neste apêndice é feita uma descrição da estrutura da API desenvolvida no âmbito desta dissertação denominada de javaOnMatlab.jar.

C.1 Classe ClientTCP

A classe ClientTCP permite criar e gerir clientes que usem *sockets* sobre o protocolo TCP/IP. É uma classe muito utilizada ao longo deste trabalho fornecendo o meio de comunicação na criação de sistemas de controlo distribuído. Exemplo disto é a *gateway* intermédia utilizada para acesso remoto a sensores e actuadores apresentada na secção 3.6.

Classe ClientTCP		
ClientTCP(String ip, int port)		
void	*help()*	Mostra modo de utilização
void	*create()*	Abre *socket*
void	*destroy()*	Destrói *socket*
boolean	*isDataAvailable()*	Verifica se chegaram dados
String	*read()*	Lê dados recebidos
void	*write(String msg)*	Escreve para o *socket*
void	*setBufSize(int BUFSIZE)*	Define tamanho do *buffer* de recepção

C.2 Classe ServerTCP

A classe ServerTCP permite a criação e gestão de servidores que usem *sockets* sobre o protocolo TCP/IP. Esta classe é utilizada frequentemente ao longo deste trabalho

na criação de sistemas de controlo distribuído. Exemplo disto é a sua utilização na *gateway* remota apresentada na secção 3.5.

Classe ServerTCP		
ServerTCP(int port)		
void	*help()*	Mostra modo de utilização
void	*create()*	Abre *socket*
void	*destroy()*	Destrói *socket*
void	*accept()*	Aceitação ligação de cliente
void	*release()*	Liberta ligação de cliente
boolean	*isDataAvailable()*	Verifica se chegaram dados
String	*read()*	Lê dados recebidos
void	*write(String msg)*	Escreve para o *socket*
void	*setBufSize(int BUFSIZE)*	Define tamanho do *buffer* de recepção

C.3 Classe spAdapter

A classe spAdapter permite uma gestão simples do acesso a portas série. É uma classe muito importante para os testes efectuados. Serve para vários propósitos no âmbito desta dissertação, tais como actuação e leitura de sensores. Exemplo disto é a sua inclusão na classe scpiAccess que permite o acesso a equipamento com suporte à norma SCPI.

Classe spAdapter		
spAdapter(String serialport, int baudrate, int timeout, String name, long sleepTime)		
void	help()	Mostra modo de utilização
void	*open()*	Abre ligação porta série
void	*close()*	Fecha ligação porta série
String	*read()*	Lê dados recebidos
void	*write(String)*	Escreve dados para porta série
boolean	*isDataAvailable()*	Verifica se chegaram dados
void	*run()*	Código da *Thread*
void	*serialEvent(SerialPortEvent event)*	Tratamento chegada de dados

C.4 Classe scpiAccess

A classe scpiAccess permite o acesso a equipamentos que suportem a norma SCPI. Esta norma é apresentada na secção 2.6. A utilização desta classe permite criar instâncias capazes de gerir acessos a múltiplos equipamentos deste tipo, de forma simplificada e eficaz. No âmbito desta dissertação esta classe foi usada para o acesso a um *datalogger* HP34970A. Esta classe não suporta, actualmente, todas as suas funcionalidades, podendo no entanto serem facilmente acrescentadas.

Classe scpiAccess	
scpiAccess(String comPort)	
void	*start()*
Inicia ligação	
void	*stop()*
Pára ligação	
boolean	*checkPresence()*
Testa presença de equipamento	
double	*readTC(String type, int slot, int channel)*
Lê termopar	
double	*readDC(int slot, int channel)*
Lê tensão	
void	*setDC(double voltage, int slot, int channel)*
Ajusta tensão	
String	*sendMsg(String msg, boolean echo)*
Manda mensagem ao equipamento	

C.5 Classe owAdapter

A classe owAdapter faz uso da 1-Wire API da Maxim. Na realidade esta pode ser vista como uma extensão da mesma. Permite ver cada barramento 1-Wire como um objecto diferente e actuar sobre este em vários aspectos, tais como, ler valores de temperatura, humidade, hora, entre outros. Foram desenvolvidos métodos de pesquisas de sensores com um grau de abstracção elevado que necessitam única e exclusivamente de referência à chave do dispositivo em causa, ou à família a que pertence. Foi, adicionalmente, criada uma subclasse desta chamada owAdapter.LCD que permite a gestão de LCDs ligados ao barramento através do uso de *switches* 1-Wire de 8 canais (DS2408). Um exemplo de aplicação desta classe é mostrado na secção 4.3.

Classe owAdapter	
owAdapter(String chip, String port)	
void	*create()*
Cria acesso a barramento	
void	*destroy()*
Destrói acesso a barramento	
void	*enterCritical*()
Entrada modo exclusivo	
void	*exitCritical*()
Saída modo exclusivo	
boolean	*checkFamilyBound(long key, int familyCode)*
Verifica laço familiar	
boolean	*checkFamilyBound(String key, int familyCode)*
Verifica laço familiar	
Vector	*getAllKeys()*
Devolve todas as chaves	
Vector	*getKeysByFamily(int code)*
Devolve todas as chaves de uma família	
OneWireContainer	*getContainerByKey(String key)*
Devolve contentor pela chave	
Vector	*getContainersByKeys(String[] keys)*
Devolve contentores pelas chaves	
Vector	*getContainersByFamily(int code)*
Devolve contentores pela família	
String	*getNameByKey(String key)*
Devolve família de um sensor pela chave	
Vector	*getAllHumidities()*
Mede humidade relativa em todos os sensores	
Vector	*getAllTemperatures()*
Mede temperatura em todos os sensores	
Vector	*getAllTimes()*
Retorna tempo em todos os RTCs	
double	*getHumidityByKey(String key)*
Devolve humidade em sensor pela chave	

int	*getResistanceByKey(String key)*
Devolve valor de resistência do potenciómetro digital pela chave	
Vector	*getSensors()*
Devolve a listagem de sensores do objecto	
int	*getSpeed()*
Devolve velocidade corrente do barramento	
void	*setSpeed()*
Define velocidade do barramento	
boolean	*getStateByKey(String key)*
Devolve estado do *switch* pela chave	
double	*getTemperatureByKey(String key)*
Devolve temperatura no sensor pela chave	
long	*getTimeByKey(String key)*
Devolve hora no RTC pela chave	
String	*getTypeByKey(String key)*
Devolve tipo de sensor pela chave	
int	*getWiperByKey(String key)*
Devolve posição do potenciómetro digital pela chave	
long	*keyToLong(String[] keys)*
Transforma chave em representação String em long	
void	*listAllDevices()*
Lista todos os sensores	
void	*setStateByKey(String key, boolean value)*
Define estado de *switch* pela chave	
void	*setTemperatureResolutionByKey(String key, double resolution)*
Define resolução de medição de temperatura em sensor pela chave	
void	*setTimeByKey(String key, long time)*
Ajusta tempo em RTC pela chave	
void	*setWiperByKey(String key, int value)*
Ajusta posição do potenciómetro digital pela chave	
void	*syncAllTimes()*
Sincroniza todos os RTC's pelo PC	

void	*syncTimeByKey(String key)*
Sincroniza tempo de RTC com o do PC pela chave	
void	*updateContainers()*
Actualiza listagem de sensores do objecto	

Classe owAdapter.LCD	
owAdapter(String key)	
void	*clearDisplay()*
Limpa o LCD	
void	*doOverdrive()*
Muda velocidade de barramento para modo *Overdrive*	
void	*fixResetMode()*
Activa o modo de controlo pelo pino RSTZ	
void	*gotoPos(int line, int pos)*
Vai para posição no LCD	
void	*initLCD()*
Inicializa LCD	
void	*returnHome()*
Volta à posição inicial	
void	*setLatchOutput(int output, boolean cmd_data)*
Fixa comando nas *latches*	
double	*waitMS(long timeToWait)*
Cria um *delay* de *timeToWait* milisegundos	
void	*writeString(String message)*
Escreve String no LCD	
void	*writeChars(byte[] bytesToWrite, int offset, int length)*
Escreve sequência de caracteres no LCD a começar no *offset* definido	

Apêndice D

Hardware desenvolvido

No âmbito desta dissertação foram desenvolvidos alguns módulos de hardware. Além daquele previamente apresentado no Apêndice A, relativamente ao controlo da electroválvula, foi ainda desenvolvido um módulo conversor entre USB e 1-Wire. Este foi baseado no integrado DS2490 da Maxim e o seu esquema eléctrico é apresentado na Figura D.1.

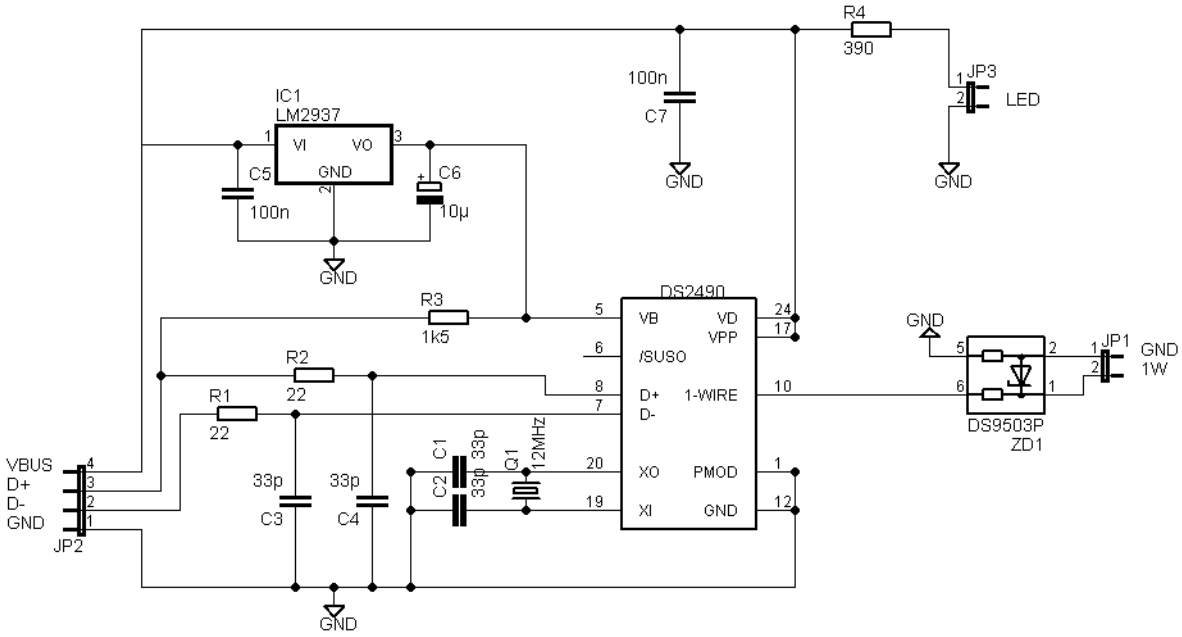
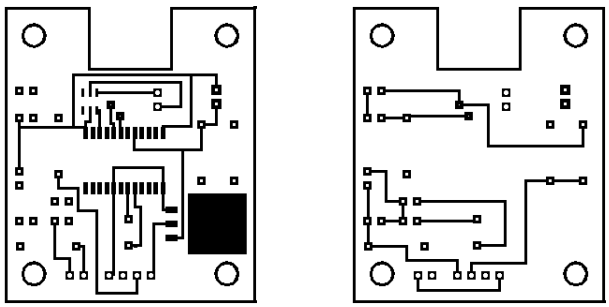


Figura D.1: Esquema eléctrico conversor 1-Wire - USB

Do projecto do conversor resultou a placa de circuito impresso apresentada na Figura D.2.



(a) Frente (b) Traseira

Figura D.2: Placa circuito impresso conversor USB - 1-Wire

O aspecto da placa finalizada é apresentado na Figura D.3.

(a) Interior (b) Exterior

Figura D.3: Conversor USB - 1-Wire

Este módulo permite ligar um barramento do tipo 1-Wire a uma interface USB. O barramento 1-Wire é ligado a este através de um conector para uma ficha do tipo RJ-45. A alimentação do conversor é feita pela interface USB e o seu tamanho é bastante reduzido (5,5 centímetros por 5 centímetros por 2 centímetros).

Bibliografia

[1] Maxim. 1-wire products - design guide, February 2009.

[2] SCPI Consortium. Standard commands for programmable instruments (scpi), 1999.

[3] Norman S. Nise. *Control Systems Engineering.* Wiley, New York, 4th edition, 2003.

[4] Ioan D. Landau and Gianluca Zito. *Digital Control Systems Design, Identification and Implementation (Communications and Control Engineering).* Springer, New York, 2006.

[5] Wu Jie. *Distributed System Design.* CRC Press, Boca Raton, FL, 1999.

[6] Charles Spurgeon. *Ethernet : The Definitive Guide.* O'Reilly & Associates, Cambridge, MA, 2000.

[7] Fei-Yue Wang and Derong Liu, editors. *Networked Control Systems-Theory and Applications.* Springer, Girona, 2008.

[8] Glenn Engel; Jerry Liu and Glen Purdy. *Java Distributed Data Acquisition and Control - User's Guide.* Agilent Technologies, 2006.

[9] Richard L. Shell and Ernest L. Hall. *Handbook of industrial automation.* CRC Press, Boca Raton, 2005.

[10] Ana Antunes. *Algoritmos de Controlo Distribuído em Sistemas Baseados em Microprocessadores.* PhD thesis, Universidade de Aveiro, 2008.

[11] Leo Motus and Michael G. Rodd. *Timing analysis of real-time software: a practical approach to the specification and design of real-time.* Pergamon, Oxford, U.K, 1994.

[12] Kevin Dooley. *Designing large-scale LANs.* O'Reilly, Beijing, 2001.

[13] Ethernet POWERLINK Standardisation Group. Ethernet powerlink: Communication profile specification, 2008.

[14] P. Pedreiras, L. Almeida, and P. Gai. The ftt-ethernet protocol: merging flexibility, timeliness and efficiency. In *Proc. 14th Euromicro Conference on Real-Time Systems*, pages 134–142, 19–21 June 2002.

[15] Kirk Reinholtz. Java will be faster than c++. *ACM SIGPLAN Notices*, 35(2):25–28, February 2000.

[16] Willie Walker; Paul Lamere and Philip Kwok. Freetts - a performance case study. Technical Report TR-2002-114, Sun MicroSystems, August 2002.

[17] James Gosling; Bill Joy; Guy Steele and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (The Java Series)*. Prentice Hall PTR, Upper Saddle River, 2005.

[18] Ken Arnold; James Gosling; David Holmes. *The Java Programming Language, Fourth Edition*. Addison Wesley Professional, August 2005.

[19] Gregory Bollella; James Gosling; Benjamin Brosgol; Peter Dibble; Steve Furr; Mark Turnbull. *The Real-time specification for Java*. Addison-Wesley, Boston, MA, 2000.

[20] Ultra-reliable 1-wire communications. Technical report, Maxim, 2003.

[21] Maxim. 1-wire® extended network standard, 2006.

[22] Extending 1-wire range with network proxies. Technical report, Maxim, 2007.

[23] Guidelines for reliable long line 1-wire® networks. Technical report, Maxim, 2008.

[24] J. P. Chancelier; F. Delebecque; C. Gomez; M. Goursat; R. Nikoukhah; and S. Steer. *Introduction à SCILAB (Collection IRIS)*. Springer, New York, 2007.

[25] Luís Farinha. Sistema de controlo distribuído baseado em ethernet. Master's thesis, Universidade de Aveiro, 2009.

[26] Alexandre Mota. *Controlo Adaptativo: um caso concreto*. PhD thesis, Universidade de Aveiro, 1992.